

```

1  /*
2  * This is the .cpp file for the SHT35D Temperature
3  * and relative humidity sensor.
4  *
5  * Part 1 of this code was retrieved online:
6  * https://github.com/closedcube/ClosedCube\_SHT31D\_Arduino
7  *
8  * Part 2 was written by MECH 45X Team 26
9  *
10 * Part 1 begins...
11 */
12
13 #include <Wire.h>
14 #include "SHT35D.h"
15
16 ClosedCube_SHT31D::ClosedCube_SHT31D()
17 {
18 }
19
20 SHT31D_ErrorCode ClosedCube_SHT31D::begin(uint8_t address) {
21     SHT31D_ErrorCode error = SHT3XD_NO_ERROR;
22     _address = address;
23     return error;
24 }
25
26 SHT31D ClosedCube_SHT31D::periodicFetchData()
27 {
28     SHT31D_ErrorCode error = writeCommand(SHT3XD_CMD_FETCH_DATA);
29     if (error == SHT3XD_NO_ERROR)
30         return readTemperatureAndHumidity();
31     else
32         returnError(error);
33 }
34
35 SHT31D_ErrorCode ClosedCube_SHT31D::periodicStop() {
36     return writeCommand(SHT3XD_CMD_STOP_PERIODIC);
37 }
38
39 SHT31D_ErrorCode ClosedCube_SHT31D::periodicStart(SHT31D_Repeatability repeatability,
40 SHT31D_Frequency frequency)
41 {
42     SHT31D_ErrorCode error;
43
44     switch (repeatability)
45     {
46     case SHT3XD_REPEATABILITY_LOW:
47         switch (frequency)
48         {
49         case SHT3XD_FREQUENCY_HZ5:
50             error = writeCommand(SHT3XD_CMD_PERIODIC_HALF_L);
51             break;
52         case SHT3XD_FREQUENCY_1HZ:
53             error = writeCommand(SHT3XD_CMD_PERIODIC_1_L);
54             break;
55         case SHT3XD_FREQUENCY_2HZ:
56             error = writeCommand(SHT3XD_CMD_PERIODIC_2_L);
57             break;
58         case SHT3XD_FREQUENCY_4HZ:
59             error = writeCommand(SHT3XD_CMD_PERIODIC_4_L);
60             break;
61         case SHT3XD_FREQUENCY_10HZ:
62             error = writeCommand(SHT3XD_CMD_PERIODIC_10_L);
63             break;
64         default:
65             error = SHT3XD_PARAM_WRONG_FREQUENCY;
66             break;
67         }
68         break;
69     case SHT3XD_REPEATABILITY_MEDIUM:

```

```

69         switch (frequency)
70         {
71             case SHT3XD_FREQUENCY_HZ5:
72                 error = writeCommand(SHT3XD_CMD_PERIODIC_HALF_M);
73                 break;
74             case SHT3XD_FREQUENCY_1HZ:
75                 error = writeCommand(SHT3XD_CMD_PERIODIC_1_M);
76                 break;
77             case SHT3XD_FREQUENCY_2HZ:
78                 error = writeCommand(SHT3XD_CMD_PERIODIC_2_M);
79                 break;
80             case SHT3XD_FREQUENCY_4HZ:
81                 error = writeCommand(SHT3XD_CMD_PERIODIC_4_M);
82                 break;
83             case SHT3XD_FREQUENCY_10HZ:
84                 error = writeCommand(SHT3XD_CMD_PERIODIC_10_M);
85                 break;
86             default:
87                 error = SHT3XD_PARAM_WRONG_FREQUENCY;
88                 break;
89         }
90         break;
91
92     case SHT3XD_REPEATABILITY_HIGH:
93         switch (frequency)
94         {
95             case SHT3XD_FREQUENCY_HZ5:
96                 error = writeCommand(SHT3XD_CMD_PERIODIC_HALF_H);
97                 break;
98             case SHT3XD_FREQUENCY_1HZ:
99                 error = writeCommand(SHT3XD_CMD_PERIODIC_1_H);
100                 break;
101             case SHT3XD_FREQUENCY_2HZ:
102                 error = writeCommand(SHT3XD_CMD_PERIODIC_2_H);
103                 break;
104             case SHT3XD_FREQUENCY_4HZ:
105                 error = writeCommand(SHT3XD_CMD_PERIODIC_4_H);
106                 break;
107             case SHT3XD_FREQUENCY_10HZ:
108                 error = writeCommand(SHT3XD_CMD_PERIODIC_10_H);
109                 break;
110             default:
111                 error = SHT3XD_PARAM_WRONG_FREQUENCY;
112                 break;
113         }
114         break;
115
116     default:
117         error = SHT3XD_PARAM_WRONG_REPEATABILITY;
118         break;
119     }
120     delay(100);
121     return error;
122 }
123
124 SHT31D ClosedCube_SHT31D::readTempAndHumidity(SHT31D_Repeatability repeatability,
125 SHT31D_Mode mode, uint8_t timeout)
126 {
127     SHT31D result;
128
129     switch (mode) {
130     case SHT3XD_MODE_CLOCK_STRETCH:
131         result = readTempAndHumidityClockStretch(repeatability);
132         break;
133     case SHT3XD_MODE_POLLING:
134         result = readTempAndHumidityPolling(repeatability, timeout);
135         break;
136     default:
137         result = returnError(SHT3XD_PARAM_WRONG_MODE);

```

```

137         break;
138     }
139     return result;
140 }
141
142
143 SHT31D ClosedCube_SHT31D::readTempAndHumidityClockStretch(SHT31D_Repeatability
repeatability)
144 {
145     SHT31D_ErrorCode error = SHT3XD_NO_ERROR;
146     SHT31D_Commands command;
147
148     switch (repeatability)
149     {
150     case SHT3XD_REPEATABILITY_LOW:
151         error = writeCommand(SHT3XD_CMD_CLOCK_STRETCH_L);
152         break;
153     case SHT3XD_REPEATABILITY_MEDIUM:
154         error = writeCommand(SHT3XD_CMD_CLOCK_STRETCH_M);
155         break;
156     case SHT3XD_REPEATABILITY_HIGH:
157         error = writeCommand(SHT3XD_CMD_CLOCK_STRETCH_H);
158         break;
159     default:
160         error = SHT3XD_PARAM_WRONG_REPEATABILITY;
161         break;
162     }
163
164     delay(50);
165
166     if (error == SHT3XD_NO_ERROR) {
167         return readTemperatureAndHumidity();
168     } else {
169         return returnError(error);
170     }
171 }
172
173
174
175 SHT31D ClosedCube_SHT31D::readTempAndHumidityPolling(SHT31D_Repeatability repeatability,
uint8_t timeout)
176 {
177     SHT31D_ErrorCode error = SHT3XD_NO_ERROR;
178     SHT31D_Commands command;
179
180     switch (repeatability)
181     {
182     case SHT3XD_REPEATABILITY_LOW:
183         error = writeCommand(SHT3XD_CMD_POLLING_L);
184         break;
185     case SHT3XD_REPEATABILITY_MEDIUM:
186         error = writeCommand(SHT3XD_CMD_POLLING_M);
187         break;
188     case SHT3XD_REPEATABILITY_HIGH:
189         error = writeCommand(SHT3XD_CMD_POLLING_H);
190         break;
191     default:
192         error = SHT3XD_PARAM_WRONG_REPEATABILITY;
193         break;
194     }
195
196     delay(50);
197
198     if (error == SHT3XD_NO_ERROR) {
199         return readTemperatureAndHumidity();
200     } else {
201         return returnError(error);
202     }
203 }

```

```

204 }
205
206 SHT31D_ClosedCube_SHT31D::readAlertHighSet() {
207     return readAlertData(SHT3XD_CMD_READ_ALR_LIMIT_HS);
208 }
209
210 SHT31D_ClosedCube_SHT31D::readAlertHighClear() {
211     return readAlertData(SHT3XD_CMD_READ_ALR_LIMIT_HC);
212 }
213
214 SHT31D_ClosedCube_SHT31D::readAlertLowSet() {
215     return readAlertData(SHT3XD_CMD_READ_ALR_LIMIT_LS);
216 }
217
218 SHT31D_ClosedCube_SHT31D::readAlertLowClear() {
219     return readAlertData(SHT3XD_CMD_READ_ALR_LIMIT_LC);
220 }
221
222
223 SHT31D_ErrorCode ClosedCube_SHT31D::writeAlertHigh(float temperatureSet, float
temperatureClear, float humiditySet, float humidityClear) {
224     SHT31D_ErrorCode error = writeAlertData(SHT3XD_CMD_WRITE_ALR_LIMIT_HS,
temperatureSet, humiditySet);
225     if (error == SHT3XD_NO_ERROR)
226         error = writeAlertData(SHT3XD_CMD_WRITE_ALR_LIMIT_HC, temperatureClear,
humidityClear);
227
228     return error;
229 }
230
231 SHT31D_ErrorCode ClosedCube_SHT31D::writeAlertLow(float temperatureClear, float
temperatureSet, float humidityClear, float humiditySet) {
232     SHT31D_ErrorCode error = writeAlertData(SHT3XD_CMD_WRITE_ALR_LIMIT_LS,
temperatureSet, humiditySet);
233     if (error == SHT3XD_NO_ERROR)
234         writeAlertData(SHT3XD_CMD_WRITE_ALR_LIMIT_LC, temperatureClear, humidityClear);
235
236     return error;
237 }
238
239 SHT31D_ErrorCode ClosedCube_SHT31D::writeAlertData(SHT31D_Commands command, float
temperature, float humidity)
240 {
241     SHT31D_ErrorCode error;
242
243     if ((humidity < 0.0) || (humidity > 100.0) || (temperature < -40.0) || (temperature
> 125.0))
244     {
245         error = SHT3XD_PARAM_WRONG_ALERT;
246     }
247     else {
248         uint16_t rawTemperature = calculateRawTemperature(temperature);
249         uint16_t rawHumidity = calculateRawHumidity(humidity);
250         uint16_t data = (rawHumidity & 0xFE00) | ((rawTemperature >> 7) & 0x001FF);
251
252         uint8_t buf[2];
253         buf[0] = data >> 8;
254         buf[1] = data & 0xFF;
255
256         uint8_t checksum = calculateCrc(buf);
257
258         Wire.beginTransaction(_address);
259         Wire.write(command >> 8);
260         Wire.write(command & 0xFF);
261         Wire.write(buf[0]);
262         Wire.write(buf[1]);
263         Wire.write(checksum);
264         return (SHT31D_ErrorCode)(-10 * Wire.endTransmission());
265     }

```

```

266
267     return error;
268 }
269
270
271 SHT31D_ErrorCode ClosedCube_SHT31D::writeCommand(SHT31D_Commands command)
272 {
273     Wire.beginTransaction(_address);
274     Wire.write(command >> 8);
275     Wire.write(command & 0xFF);
276     return (SHT31D_ErrorCode)(-10 * Wire.endTransmission());
277 }
278
279 SHT31D_ErrorCode ClosedCube_SHT31D::softReset() {
280     return writeCommand(SHT3XD_CMD_SOFT_RESET);
281 }
282
283 SHT31D_ErrorCode ClosedCube_SHT31D::generalCallReset() {
284     Wire.beginTransaction(0x0);
285     Wire.write(0x06);
286     return (SHT31D_ErrorCode)(-10 * Wire.endTransmission());
287 }
288
289 SHT31D_ErrorCode ClosedCube_SHT31D::heaterEnable() {
290     return writeCommand(SHT3XD_CMD_HEATER_ENABLE);
291 }
292
293 SHT31D_ErrorCode ClosedCube_SHT31D::heaterDisable() {
294     return writeCommand(SHT3XD_CMD_HEATER_DISABLE);
295 }
296
297 SHT31D_ErrorCode ClosedCube_SHT31D::artEnable() {
298     return writeCommand(SHT3XD_CMD_ART);
299 }
300
301
302 uint32_t ClosedCube_SHT31D::readSerialNumber()
303 {
304     uint32_t result = SHT3XD_NO_ERROR;
305     uint16_t buf[2];
306
307     if (writeCommand(SHT3XD_CMD_READ_SERIAL_NUMBER) == SHT3XD_NO_ERROR) {
308         if (read(buf, 2) == SHT3XD_NO_ERROR) {
309             result = (buf[0] << 16) | buf[1];
310         }
311     }
312
313     return result;
314 }
315
316 SHT31D_RegisterStatus ClosedCube_SHT31D::readStatusRegister()
317 {
318     SHT31D_RegisterStatus result;
319
320     SHT31D_ErrorCode error = writeCommand(SHT3XD_CMD_READ_STATUS);
321     if (error == SHT3XD_NO_ERROR)
322         error = read(&result.rawData, 1);
323
324     return result;
325 }
326
327 SHT31D_ErrorCode ClosedCube_SHT31D::clearAll() {
328     return writeCommand(SHT3XD_CMD_CLEAR_STATUS);
329 }
330
331
332 SHT31D ClosedCube_SHT31D::readTemperatureAndHumidity()
333 {
334     SHT31D result;

```

```

335
336     result.t = 0;
337     result.rh = 0;
338
339     SHT31D_ErrorCode error;
340     uint16_t buf[2];
341
342     if (error == SHT3XD_NO_ERROR)
343         error = read(buf, 2);
344
345     if (error == SHT3XD_NO_ERROR) {
346         result.t = calculateTemperature(buf[0]);
347         result.rh = calculateHumidity(buf[1]);
348     }
349     result.error = error;
350
351     return result;
352 }
353
354 SHT31D_ClosedCube_SHT31D::readAlertData(SHT31D_Commands command)
355 {
356     SHT31D result;
357
358     result.t = 0;
359     result.rh = 0;
360
361     SHT31D_ErrorCode error;
362     uint16_t buf[1];
363
364     error = writeCommand(command);
365
366     if (error == SHT3XD_NO_ERROR)
367         error = read(buf, 1);
368
369     if (error == SHT3XD_NO_ERROR) {
370         result.rh = calculateHumidity(buf[0] << 7);
371         result.t = calculateTemperature(buf[0] & 0xFE00);
372     }
373
374     result.error = error;
375
376     return result;
377 }
378
379 SHT31D_ErrorCode ClosedCube_SHT31D::read(uint16_t* data, uint8_t numOfPair)
380 {
381     uint8_t checksum;
382     char buf[2];
383     uint8_t buffer[2];
384
385
386     const uint8_t numOfBytes = numOfPair * 3;
387     Wire.requestFrom(_address, numOfBytes);
388
389     int counter = 0;
390
391     for (counter = 0; counter < numOfPair; counter++) {
392         Wire.readBytes(buf, 2);
393         checksum = Wire.read();
394
395         for (int i = 0; i < 2; i++){buffer[i] = uint8_t(buf[i]);}
396
397
398         if (checkCrc(buffer, checksum) != 0)
399             return SHT3XD_CRC_ERROR;
400
401         data[counter] = (buf[0] << 8) | buf[1];
402     }
403

```

```

404     return SHT3XD_NO_ERROR;
405 }
406
407
408 uint8_t ClosedCube_SHT31D::checkCrc(uint8_t data[], uint8_t checksum)
409 {
410     return calculateCrc(data) != checksum;
411 }
412
413 float ClosedCube_SHT31D::calculateTemperature(uint16_t rawValue)
414 {
415     return 175.0f * (float)rawValue / 65535.0f - 45.0f;
416 }
417
418
419 float ClosedCube_SHT31D::calculateHumidity(uint16_t rawValue)
420 {
421     return 100.0f * rawValue / 65535.0f;
422 }
423
424 uint16_t ClosedCube_SHT31D::calculateRaWTemperature(float value)
425 {
426     return (value + 45.0f) / 175.0f * 65535.0f;
427 }
428
429 uint16_t ClosedCube_SHT31D::calculateRawHumidity(float value)
430 {
431     return value / 100.0f * 65535.0f;
432 }
433
434 uint8_t ClosedCube_SHT31D::calculateCrc(uint8_t data[])
435 {
436     uint8_t bit;
437     uint8_t crc = 0xFF;
438     uint8_t dataCounter = 0;
439
440     for (; dataCounter < 2; dataCounter++) {
441         crc ^= (data[dataCounter]);
442         for (bit = 8; bit > 0; --bit) {
443             if (crc & 0x80){crc = (crc << 1) ^ 0x131;}
444             else {crc = (crc << 1);}
445         }
446     }
447
448     return crc;
449 }
450
451 SHT31D ClosedCube_SHT31D::returnError(SHT31D_ErrorCode error) {
452     SHT31D result;
453     result.t = 0;
454     result.rh = 0;
455     result.error = error;
456     return result;
457 }
458
459 /*
460  * Part 2: Code Written by team 26
461  * Team 26 understands this code,
462  * therefore it is properly commented.
463  */
464 bool ClosedCube_SHT31D::start_sht(void) {
465     /*
466     * Start sequence for SHT35D
467     * Return true: sensor was sucessfully started
468     * Return false: sensor was not started
469     * Try to read from sensor
470     * If no error, return true
471     * Else return false
472     */

```

```

473 Serial.println("Trying to start SHT sensor...");
474 delay(500);
475 begin(ADDR_SHT); // I2C address: 0x44 or 0x45
476 Serial.print("Serial #");
477 Serial.println(readSerialNumber());
478 delay(500);
479
480 if (periodicStart(SHT3XD_REPEATABILITY_HIGH, SHT3XD_FREQUENCY_10HZ) !=
SHT3XD_NO_ERROR) {
481     Serial.println("[ERROR] Cannot start periodic mode");
482     return false;
483 }
484 else {
485     Serial.println("Successfully started SHT sensor!");
486     return true;
487 }
488 }
489
490 void ClosedCube_SHT31D::run_sht(void) {
491     /*
492     * Run SHT sensor
493     * start read_count from 1
494     * is_average_taken is false until average is taken
495     * take reading from sht until enough values are read to take an average
496     */
497     is_average_taken = false;
498     read_count = 1;
499     while(!is_average_taken) {read_sht();}
500 }
501
502 SHT31D ClosedCube_SHT31D::read_sht(void) {
503     /*
504     * Read from SHT35D, and assign values to my_result
505     * print results
506     * save results to buffer
507     * calculate average if enough values have been read
508     */
509     SHT31D my_result = periodicFetchData();
510     printResult("Periodic Mode", my_result);
511     save_to_buffer(my_result);
512     calculate_average();
513     delay(500);
514 }
515
516 SHT31D ClosedCube_SHT31D::save_to_buffer(SHT31D result) {
517     /*
518     * Save current t and rh readings to their respective buffers
519     *
520     * if no error and the number of readings is less than the max
521     * then save values
522     *
523     * else -> report error, do not save any values
524     */
525     if (result.error == SHT3XD_NO_ERROR && read_count <= MAX_READ_COUNT) {
526         t_buf[read_count - 1] = result.t;
527         rh_buf[read_count - 1] = result.rh;
528         read_count++;
529     }
530     else {
531         Serial.print("[ERROR] Code #");
532         Serial.println(result.error);
533     }
534 }
535
536 SHT31D ClosedCube_SHT31D::printResult(String text, SHT31D result) {
537     /*
538     * Prints current reading if no error and not exceeded max count
539     * else print error message
540     */

```



```

541     if (result.error == SHT3XD_NO_ERROR && read_count <= MAX_READ_COUNT) {
542         Serial.print(text);
543         Serial.print(" Reading #");
544         Serial.print(read_count);
545         Serial.print(": T=");
546         Serial.print(result.t);
547         Serial.print("C, RH=");
548         Serial.print(result.rh);
549         Serial.println("%");
550     }
551     else {
552         Serial.print(text);
553         Serial.print(": [ERROR] Code #");
554         Serial.println(result.error);
555     }
556 }
557
558 void ClosedCube_SHT31D::calculate_average(void) {
559     /*
560     * Calculate average if enough values have been read
561     * assign t ave to t_average
562     * assign rh ave to rh_average
563     * change is_average_taken to true so that while loop will exit
564     */
565     if( read_count > MAX_READ_COUNT ) {
566         t_average = 0.00;
567         rh_average = 0.00;
568         for(int k = 0; k < MAX_READ_COUNT; k++) {
569             t_average += t_buf[k];
570             rh_average += rh_buf[k];
571         }
572         t_average = t_average / MAX_READ_COUNT;
573         rh_average = rh_average / MAX_READ_COUNT;
574
575         delay(500);
576         Serial.println("-----");
577         Serial.println("SHT Sensor Average Readings");
578         Serial.println("-----");
579         Serial.print("SHT T Average: ");
580         Serial.println(t_average);
581         Serial.print("SHT RH Average: ");
582         Serial.println(rh_average);
583         is_average_taken = true;
584     }
585 }
586
587 // getter function to get average temperature reading
588 float ClosedCube_SHT31D::get_t_ave(void) {return t_average;}
589
590 // getter function to get average relative humidity reading
591 float ClosedCube_SHT31D::get_rh_ave(void) {return rh_average;}
592

```