```
 1: /* MonoTronic.c
 2: *  Team DERP
 3: *  Automatic Air Quality Control */
 4:
 5: #include<stdbool.h>
 6:
 7: // Using LCD display to show mode of operation of AC, can be replaced by motor driv
    r control
 8: // Define pins for LCD interface
 9: #define LCD_Data_Dir DDRB               // Define LCD data port direction
10: #define LCD_Command_Dir DDRC            // Define LCD command port direction registe
    er
11: #define LCD_Data_Port PORTB             // Define LCD data port
12: #define LCD_Command_Port PORTC          // Define LCD data port
13: #define RS PORTC0                       // Define Register Select (data/command reg.
    .)pin
14: #define RW PORTC1                       // Define Read/Write signal pin
15: #define EN PORTC2
16:
17: // Define pins for temperature and gas sensors
18: #define CO_in_pin PORTA2                    // CO sensor inside cabin
19: #define CO_out_pin PORTA1                   // CO sensor outside cabin
20: #define CO2_in_pin PORTA4                   // CO2 sensor inside cabin
21: #define CO2_out_pin PORTA3                  // CO2 sensor outside cabin
22:
23:
24: // Declare variables
25: int CO_in;       // CO sensor value inside cabin
26: int CO_out;      // CO sensor value outside cabin
27: int CO2_in;      // CO2 sensor value inside cabin
28: int CO2_out;     // CO2 sensor value outside cabin
29: int occupancy;      // Number of occupants inside car
30: int AC_mode;        // current AC mode, 0 - recirculation, 1 - fresh air, 2 - AC off
31: int CO_flag;        // To check increase of CO
32: bool flag;         // To check when case 1 and case 2 were detected for CO
33: bool CO2_check = false;   // To decide which CO2 loop to enter
34: int CO2_occu;       // Threshold CO2 based on occupancy
35: char str1[2];
36: char str2[2];
37: char str3[2];
38: char str4[2];
39:
40:
41:
42: // Declare functions
43: void CO_control();              // To control CO concentration
44: void CO2_control();             // To control CO2 contentration
45: int CO2_occu_get(int);          // Return threshold CO2 based on occupancy
46: int check_occupancy();          // Get the number of occupants inside car
47: float get_CO_in();              // Get CO inside cabin in ppm
48: float get_CO_out();             // Get CO outside cabin in ppm
49: float get_CO2_in();             // Get CO2 inside cabin in ppm
50: float get_CO2_out();            // Get CO2 outside cabin in ppm
51: void send_data(int, int , int, int);  // Send sensor data through UART
52:
53: // 16 x 2 bit LCD related functions
```

```
 54: void LCD_Command(unsigned char);
 55: void LCD_Char (unsigned char);
 56: void LCD_start (void);
 57: void LCD_String (char);
 58: void LCD_String_xy (char, char, char);
 59: void LCD_Clear();
 60: void recirculation_mode();       // Print recirculation mode on LCD
 61: void fresh_air_mode();           // Print fresh air mode on LCD
 62: void AC_off();                   // Print AC off on LCD
 63:
 64:
 65: void main(){
 66:
 67:      // Set I/O ports
 68:      DDRA = 0x00;
 69:      DDRB = 0xFF;
 70:      DDRC = 0xFF;
 71:      DDRD = 0x00;
 72:
 73:
 74:      ADC_Init();           // Initialize ADC
 75:      LCD_start();          // Initialize LCD
 76:      UART1_Init(9600);     // Initialize UART communication with baud rate of 9600
 77:      Delay_ms(10);
 78:      recirculation_mode();
 79:      AC_mode = 0;
 80:      Delay_ms(10);
 81:
 82:      flag = false;
 83:
 84:      // Main control loop
 85:      while(1){
 86:               CO_in = get_CO_in();
 87:               CO_out = get_CO_out();
 88:               CO2_in = get_CO2_in();
 89:               CO2_out = get_CO2_out();
 90:               Delay_ms(50);
 91:               if(CO_in >= 1000 || CO2_in >= 10000)
 92:               {
 93:                        AC_off();
 94:                        AC_mode = 2;
 95:               }
 96:               if(CO_in >= 30)
 97:               {
 98:                        CO_control();
 99:               }
100:               if(CO_in < 30)
101:               {
102:                        if(CO2_in >= 1300)
103:                        {
104:                                 CO2_control();
105:                        }
106:                        occupancy = check_occupancy();
107:                        CO2_occu =  CO2_occu_get(occupancy);
108:                        if(CO2_in < CO2_occu && CO2_check == true)
109:                        {
110:                                  recirculation_mode();       // Switch back to recir
    rculation once below threshold
```

```
111:                                    AC_mode = 0;
112:                                    CO2_check = false;
113:                            }
114:                }
115:                send_data(CO_in, CO_out, CO2_in, CO2_out);
116:                Delay_ms(1000);
117:
118:       }
119: }
120:
121: // Function to control Carbon Monoxide(CO)
122: void CO_control(){
123:       if(CO_in - CO_out >= 10)       // case 1
124:       {
125:                if(AC_mode == 0)      // If AC was in recirculation, set to fresh air
126:                {
127:                          fresh_air_mode();
128:                          AC_mode = 1;
129:                }
130:                if(AC_mode == 1)     // If AC was in fresh air, check for increase
131:                {
132:                          if(flag == false)     // First time case 1 detected in fresh
      air mode
133:                                    {
134:                                       CO_flag = CO_in;
135:                                       flag = true;
136:                                    }
137:                          if(CO_in - CO_flag > 20)     // If CO increased by 20 ppm sin
     nce this case detected so turn AC off
138:                          {
139:                                    AC_off();
140:                                    AC_mode = 2;
141:                                    CO_flag = 0;
142:                                    flag = false;
143:                                    Delay_ms(2);
144:                          }
145:                }
146:       }
147:       if(CO_out - CO_in >= 10)       // case 2
148:       {
149:                 if(AC_mode == 1)      // If AC was in fresh air, set to recirculation
150:                {
151:                          recirculation_mode();
152:                          AC_mode = 0;
153:                }
154:                if(AC_mode == 0)       // If AC was in recirculation, check for increas
     se
155:                {
156:                          if(flag == false)        // First time this case detected
157:                                    {
158:                                       CO_flag = CO_in;
159:                                       flag = ~flag;
160:                                    }
161:                          if((CO_in - CO_flag) > 20)  // If CO increased by 20 ppm sin
     nce this case detected so turn AC off
162:                          {
163:                                    AC_off();
```

```
164:                                    CO_flag = 0;
165:                                    flag = ~flag;
166:                                    AC_mode = 2;
167:                            }
168:                }
169:
170:      }
171: }
172:
173: // Function to control Carbon Dioxide(CO2)
174: void CO2_control()
175: {
176:      if(AC_mode == 0)                  // If AC was in recirculation mode
177:      {
178:                fresh_air_mode();       // Set to fresh air mode
179:                AC_mode = 1;
180:                occupancy = check_occupancy();
181:                Delay_ms(10);
182:                CO2_occu =  CO2_occu_get(occupancy);   // Get threshold based on occu
    upancy
183:                CO2_check = true;
184:      }
185:      if(AC_mode == 1 && CO2_check == false)      // If fresh air mode while CO2 high
     outside
186:      {
187:                if(CO2_out >= 2000)               // CO2 outside high
188:                           {
189:                              recirculation_mode();
190:                              AC_mode = 0;
191:                              while(CO2_out > 1300)
192:                              {
193:                                          CO2_out = get_CO2_out();
194:                                          send_data(CO_in, CO_out, CO2_in, CO2_out
    );
195:                                          Delay_ms(1000);
196:
197:                              }
198:                              fresh_air_mode();   // Set back to fresh air once CO2 o
    outside is low
199:                              AC_mode = 1;
200:                           }
201:      }
202: }
203:
204: // Get threshold CO2 value based on occupancy
205: int CO2_occu_get(int occupancy)
206: {
207:     switch(occupancy)
208:    {
209:               case 1:
210:                    return 550; break;
211:               case 2:
212:                    return 700; break;
213:               case 3:
214:                    return 850; break;
215:               case 4:
216:                    return 1000; break;
```

```
217:              case 5:
218:                   return 1150; break;
219:    }
220: }
221:
222:
223: // Get gas sensor values
224: float get_CO_in(){
225:      return (ADC_Read(CO_in_pin)*0.48828125);
226: }
227:
228: float get_CO_out(){
229:      return (ADC_Read(CO_out_pin)*0.48828125);
230: }
231:
232: float get_CO2_in(){
233:      return (ADC_Read(CO2_in_pin)*0.48828125*100);
234: }
235:
236: float get_CO2_out(){
237:      return (ADC_Read(CO2_out_pin)*0.48828125*100);
238: }
239:
240: int check_occupancy()            // Count occupancy inside car
241: {
242:     int count = 1;               // Assumed driver is present
243:     if(PIND.B2 == 1)
244:                 count = count + 1;
245:     if(PIND.B3 == 1)
246:                 count = count + 1;
247:     if(PIND.B4 == 1)
248:                 count = count + 1;
249:     if(PIND.B5 == 1)
250:                 count = count + 1;
251:     return count;
252: }
253:
254: // Functions for LCD
255: void LCD_Command(unsigned char cmnd)
256: {
257:         LCD_Data_Port= cmnd;
258:         LCD_Command_Port &= ~(1<<RS);
259:         LCD_Command_Port &= ~(1<<RW);
260:         LCD_Command_Port |= (1<<EN);
261:         Delay_us(1);
262:         LCD_Command_Port &= ~(1<<EN);
263:         Delay_ms(3);
264: }
265:
266: void LCD_Char (unsigned char char_data)        // LCD data write function
267: {
268:         LCD_Data_Port= char_data;
269:         LCD_Command_Port |= (1<<RS);
270:         LCD_Command_Port &= ~(1<<RW);
271:         LCD_Command_Port |= (1<<EN);
272:         Delay_us(1);
273:         LCD_Command_Port &= ~(1<<EN);
```

```
274:         Delay_ms(1);
275: }
276:
277: void LCD_start (void)                          // LCD Initialize function
278: {
279:         LCD_Command_Dir = 0xFF;               // Make LCD command port direction as o/p
     p
280:         LCD_Data_Dir = 0xFF;                  // Make LCD data port direction as o/p
281:         Delay_ms(20);                         // LCD Power ON delay always >15ms
282:         LCD_Command (0x38);                   // Initialization of 16X2 LCD in 8bit mod
     de
283:         LCD_Command (0x0C);                   // Display ON Cursor OFF
284:         LCD_Command (0x06);                   // Auto Increment cursor
285:         LCD_Command (0x01);                   // Clear display
286:         LCD_Command (0x80);                   // Cursor at home position
287: }
288:
289: void LCD_String (char *str)                  // Send string to LCD function
290: {
291:         int i;
292:         for(i=0;str[i]!=0;i++)                // Send each char of string till the NU
     ULL
293:         {
294:                 LCD_Char (str[i]);
295:         }
296: }
297:
298: void LCD_String_xy (char row, char pos, char *str)  // Send string to LCD with xy pos
     sition
299: {
300:         if (row == 0 && pos<16)
301:         LCD_Command((pos & 0x0F)|0x80);        //Command of first row and required po
     osition<16
302:         else if (row == 1 && pos<16)
303:         LCD_Command((pos & 0x0F)|0xC0);        // Command of first row and required p
     position<16
304:         LCD_String(str);                       // Call LCD string function
305: }
306:
307: void LCD_Clear()                              // Clear LCD screen
308: {
309:         LCD_Command (0x01);                   // clear display
310:         LCD_Command (0x80);                   // cursor at home position
311: }
312:
313: void recirculation_mode()                     // Print "Recirculation Mode" on LCD
314: {
315:      LCD_Clear();
316:      Delay_ms(10);
317:      LCD_String("Recirculation");
318:      LCD_Command(0xC0);
319:      LCD_String("Mode");
320: }
321:
322: void fresh_air_mode()                         // Print "Fresh Air Mode" on LCD
323: {
324:       LCD_Clear();
```

```
325:       Delay_ms(10);
326:       LCD_String("Fresh Air");
327:       LCD_Command(0xC0);
328:       LCD_String("Mode");
329: }
330:
331: void AC_off()                       // Print "AC off" on LCD
332: {
333:       LCD_Clear();
334:       Delay_ms(10);
335:       LCD_String("AC");
336:       LCD_Command(0xC0);
337:       LCD_String("off");
338: }
339:
340: // Send sensor values through UART
341: void send_data(int co_in,int co_out,int co2_in,int co2_out) {
342:
343:    Delay_ms(10);
344:    UART1_Write(13);
345:    floatToStr(co_in,str1);
346:    UART1_write_Text("CO_in:");
347:    UART1_write_Text(str1);
348:    UART1_Write(13);
349:
350:    floatToStr(co_out,str2);
351:    UART1_write_Text("CO_out:");
352:    UART1_write_Text(str2);
353:    UART1_Write(13);
354:
355:    floatToStr(co2_in,str3);
356:    UART1_write_Text("CO2_in:");
357:    UART1_write_Text(str3);
358:    UART1_Write(13);
359:
360:    floatToStr(co2_out,str4);
361:    UART1_write_Text("CO2_out:");
362:    UART1_write_Text(str4);
363:    UART1_Write(13);
364: }
```