```
001 //Власов Роман Евгеньевич
002 //группа 250541
003 //Микропроцессорное устройство распознавания речи
004
005 // Определение макроса uspeech_h
006 #define uspeech_h
007 // Подключение заголовочного файла Arduino.h для доступа к функциям
    платформы Arduino
008 #include "Arduino.h"
009 // Подключение библиотеки для работы со строками
010 #include <string.h>
011 // Подключение математической библиотеки для использования математических
    функций
012 #include <math.h>
013 // Подключение стандартной библиотеки для общего функционала
014 #include <stdlib.h>
015
016 // Определение порога тишины
017 #define SILENCE 2000
018 // Определение константы F_DETECTION для обнаружения фонемы /f/
019 #define F_DETECTION 3
020 // Определение константы F_CONSTANT (порог для /f/)
021 #define F_CONSTANT 350
022 // Определение константы MAX_PLOSIVETIME (максимальное время для
    определения взрывного звука)
023 #define MAX_PLOSIVETIME 1000
024 // Определение константы PROCESS_SKEWNESS_TIME (период анализа данных)
025 #define PROCESS_SKEWNESS_TIME 15
026
027 //Класс для распознавания звука
028 class signal {
029 public:
030 // Объявление массива для хранения аудиоданных (буфер из 32 выборок)
031     int arr[32];
032 // Объявление переменной для средней мощности сигнала
033     int avgPower;
034 // Объявление тестового коэффициента для отладки
035     int testCoeff;
036 // Объявление минимального значения громкости, после которого сигнал
    считается готовым к распознанию
037     int minVolume;
038 // Объявление порога для распознавания фонемы /f/ (настраиваемый)
039     int fconstant;
040 // Объявление порога для распознавания фонем, таких как /ee/ или /i/
041     int econstant;
042 //Объявление порога для распознавания фонем, таких как /a/, /o/, /r/, /l/
043     int aconstant;
044 // Объявление порога для распознавания фонем, таких как /z/, /v/ или /w/
045     int vconstant;
046 // Объявление порога для распознавания фонем, таких как /sh/ или /ch/
    (значения выше - фонема /s/)
047     int shconstant;
048 // Объявление флага, разрешающего распознавание фонемы /f/
049     bool f_enabled;
050 // Объявление коэффициента усиления для регулировки чувствительности
051     int amplificationFactor;
052 // Объявление порога мощности микрофона (значения ниже игнорируются)
053     int micPowerThreshold;
054 // Объявление масштабного коэффициента для входного сигнала
055     int scale;
056 // Объявление переменной для хранения распознанной фонемы (символ)
057     char phoneme;
058 // Объявление конструктора класса, принимающего номер порта
```

```
059     signal(int port);
060 // Объявление переменной для хранения мощности микрофона
061     int micPower;
062 // Объявление метода для выборки звука
063     void sample();
064 // Объявление метода для определения максимальной мощности сигнала
065     unsigned int maxPower();
066 // Объявление метода для вычисления общей мощности сигнала
067     unsigned int power();
068 // Объявление метода для вычисления отношения сигнал/шум (SNR)
069     int snr(int power);
070 //Объявление метода для калибровки микрофона (определение фонового уровня)
071     void calibrate();
072 // Объявление метода для распознавания фонемы (возврат символа)
073     char getPhoneme();
074 // Объявление переменной для хранения значения калибровки (средний уровень
    фонового шума)
075     int calib;
076 private:
077 // Объявление переменной для хранения номера порта, к которому подключён
    микрофон
078     int pin;
079 // Объявление переменной для хранения времени (миллисекунды) – может
    использоваться для отладки или тайминга
080     int mil;
081 // Объявление переменной для хранения позиции в массиве с максимальной
    амплитудой
082     int maxPos;
083     // Объявление флага, указывающего на наличие тишины
084     bool silence;
085 // Объявление массива для хранения истории коэффициентов (используемого
    для фильтрации шума)
086     unsigned int overview[7];
087 // Объявление приватного метода для вычисления «сложности» сигнала
    (отношение суммы модулей производных к мощности)
088     unsigned int complexity(int power);
089 };
090
091 //Класс для накопления статистических показателей
092 class statCollector {
093 public:
094     // Объявление переменных для количества выборок, среднего значения и
    моментов 2-го, 3-го и 4-го порядка
095     int n, mean, M2, M3, M4;
096 // Объявление конструктора класса
097     statCollector();
098 // Объявление метода для вычисления куртоза (остроты распределения)
099     int kurtosis();
100 // Объявление метода для вычисления асимметрии распределения
101     int skew();
102 // Объявление метода для получения среднего значения
103     int _mean();
104 //Объявление метода для получения стандартного отклонения(возвращается M2)
105     int stdev();
106 // Объявление метода для накопления статистических данных (обновление
    моментов) с новым значением x
107     void collect(int x);
108 };
109
110 //Класс для аккумуляции фонем (слогов)
111 class syllable {
112 public:
113 // Объявление аккумуляторов для счёта появлений фонем: f, e, o, s, h, v
```

```
114     int f, e, o, s, h, v;
115 // Объявление переменных для хранения максимальных позиций (или
    показателей) для каждой фонемы
116     int maxf, maxe, maxo, maxs, maxh, maxv;
117 // Объявление переменных для модальности каждой фонемы (индикатор наличия
    двух пиков в распределении)
118     int modalityf, modalitye, modalityo, modalitys, modalityh, modalityv;
119 // Объявление переменной для хранения длины произнесённого слога
    (количество фонем)
120     int length;
121 // Объявление счётчика взрывных звуков (плозивов)
122     int plosiveCount;
123 // Объявление конструктора класса syllable
124     syllable();
125 // Объявление метода для сброса аккумуляторов (вызывается при обнаружении
    тишины)
126     void reset();
127 // Объявление метода для классификации символа (фонемы) с обновлением
    аккумуляторов в зависимости от входного символа
128     void classify(char c);
129 // Объявление метода для возврата аккумуляторов в виде указателя на массив
    int
130     int* tointptr();
131 // Объявление метода для отладочного вывода накопленных данных на Arduino
132     void debugPrint();
133 // Объявление метода для расчёта «расстояния» (схожести) между двумя
    слогами
134     void distance(syllable s);
135 private:
136 // Объявление временных аккумуляторов для промежуточного подсчёта фонем
137     char cf, ce, co, cs, ch, cv;
138 // Объявление переменных для хранения предыдущих значений временных
    аккумуляторов (для вычисления пиков и модальностей)
139     char prevf, preve, prevo, prevs, prevh, prevv;
140 // Объявление переменной для хранения текущего пика (фонемы) и флага
    ожидания пробела (для разделения слогов)
141     char currPeak, expectSp;
142 // Объявление переменной для хранения времени последнего обновления
    (определение интервала между слогами)
143     unsigned long lastTime;
144 };
145
146 #include "uspeech.h"
147
148 // Реализация метода распознавания фонемы getPhoneme класса signal
149 char signal::getPhoneme() {
150 // Выполнение выборки звуковых данных
151     sample();
152 // Вычисление суммарной мощности (энергии) сигнала
153     unsigned int pp = power();
154
155 // Сравнение вычисленной мощности с пороговым значением тишины
156     if (pp > SILENCE) {
157 // Вычисление "сложности" сигнала
158         int k = complexity(pp);
159 // Сдвиг значений в истории коэффициентов (низкочастотная фильтрация)
160         overview[6] = overview[5];
161         overview[5] = overview[4];
162         overview[4] = overview[3];
163         overview[3] = overview[2];
164         overview[2] = overview[1];
165         overview[1] = overview[0];
166 // Запись нового значения сложности в начало массива истории
```

```
167         overview[0] = k;
168 // Инициализация переменной для расчёта среднего коэффициента
169         int coeff = 0;
170 // Итерация по элементам массива для суммирования значений
171         for (uint8_t f = 0; f < 6; f++) {
172             coeff += overview[f];
173         }
174 // Вычисление среднего коэффициента фильтрации
175         coeff /= 7;
176 // Расчёт мощности микрофона с экспоненциальным сглаживанием
177         micPower = 0.05 * maxPower() + (1 - 0.05) * micPower;
178 // Сохранение вычисленного коэффициента для отладки
179         testCoeff = coeff;
180 // Классификация фонемы на основе среднего коэффициента
181         if (coeff < econstant) {
182 // Выбор фонемы 'e'
183             phoneme = 'e';
184         } else if (coeff < aconstant) {
185 // Выбор фонемы 'o'
186             phoneme = 'o';
187         } else if (coeff < vconstant) {
188 // Выбор фонемы 'v'
189             phoneme = 'v';
190         } else if (coeff < shconstant) {
191 // Выбор фонемы 'h'
192             phoneme = 'h';
193         } else {
194 // Выбор фонемы 's'
195             phoneme = 's';
196         }
197 // Проверка разрешения распознавания фонемы /f/
198         if (f_enabled) {
199 // Сравнение мощности микрофона с порогом для фонемы /f/
200             if (micPower > fconstant) {
201                 // Возврат фонемы 'f'
202                 return 'f';
203             }
204         }
205 // Возврат распознанной фонемы
206         return phoneme;
207     }
208     else {
209 // Обнуление мощности микрофона при отсутствии звука
210         micPower = 0;
211 // Обнуление тестового коэффициента
212         testCoeff = 0;
213 // Возврат пробела (отсутствие звука)
214         return ' ';
215     }
216 }
217
218 #include "uspeech.h"
219
220 // Реализация конструктора класса signal
221 signal::signal(int port) {
222 // Присвоение номера порта для микрофона
223     pin = port;
224 // Инициализация порога для фонемы /f/ значением константы F_CONSTANT
225     fconstant = F_CONSTANT;
226 // Установка порога для фонем 'e'
227     econstant = 2;
228 // Установка порога для фонем 'o'
229     aconstant = 4;
```

```
230 // Установка порога для фонем 'v'
231     vconstant = 6;
232 // Установка порога для фонем 'h'
233     shconstant = 10;
234 // Установка коэффициента усиления для вычисления сложности сигнала
235     amplificationFactor = 10;
236 // Задание порога, ниже которого мощность микрофона считается слишком
    низкой
237     micPowerThreshold = 50;
238 // Установка масштаба для входного сигнала
239     scale = 1;
240 }
241
242 // Реализация метода калибровки микрофона на основе усреднения фонового
    уровня шума
243 void signal::calibrate(){
244  // Инициализация переменной калибровки нулем
245     calib = 0;
246  // Инициализация переменной для суммирования измерений
247     uint32_t samp = 0;
248  // Цикл для сбора 10 000 выборок фонового шума
249     for (uint16_t ind = 0; ind < 10000; ind++){
250 // Считывание значения с аналового входа, масштабирование и накопление
251         samp += analogRead(pin) * scale;
252     }
253 // Вычисление среднего значения фонового шума
254     calib = samp / 10000;
255 }
256
257 // Реализация метода выборки звукового сигнала с вычитанием калибровочного
    значения
258 void signal::sample(){
259 // Инициализация счётчика выборки
260     int i = 0;
261 // Цикл для сбора 32 значений с микрофона
262     while (i < 32){
263 // Считывание значения с аналога, масштабирование, вычитание калибровки и
    запись в массив
264         arr[i] = (analogRead(pin) * scale - calib);
265 // Инкремент счётчика
266         i++;
267     }
268 }
269
270 // Реализация метода для вычисления общей мощности сигнала
271 unsigned int signal::power(){
272 // Инициализация переменной для накопления суммы модулей значений сигнала
273     unsigned int j = 0;
274 // Инициализация счётчика
275     uint8_t i = 0;
276 // Цикл по всем элементам массива с выборками
277     while (i < 32){
278 // Прибавление абсолютного значения текущей выборки к сумме
279         j += abs(arr[i]);
280 // Инкремент счётчика
281         i++;
282     }
283 // Возврат суммарной мощности сигнала
284     return j;
285 }
286
287 // Реализация метода для вычисления «сложности» сигнала (отношение суммы
    модулей разностей соседних значений к мощности)
```

```
unsigned int signal::complexity(int power){
// Инициализация переменной для накопления суммы модулей разностей соседних выборок
    unsigned int j = 0;
// Установка счётчика с началом со второго элемента массива
    uint8_t i = 1;
// Цикл по элементам массива, начиная со второго значения
    while (i < 32){
// Прибавление модуля разности текущей и предыдущей выборки
        j += abs(arr[i] - arr[i - 1]);
// Инкремент счётчика
        i++;
    }
// Вычисление и возврат значения «сложности» сигнала с учётом коэффициента усиления
    return (j * amplificationFactor) / power;
}

// Реализация метода для определения максимальной амплитуды сигнала
unsigned int signal::maxPower() {
// Инициализация счётчика для обхода массива
    int i = 0;
// Инициализация переменной для хранения максимальной амплитуды
    unsigned int max = 0;
// Цикл по всем 32 выборкам сигнала
    while (i < 32){
// Сравнение текущего максимального значения с модулем текущей выборки
        if (max < abs(arr[i])){
// Обновление максимального значения амплитуды
            max = abs(arr[i]);
// Запись позиции, на которой зафиксирован максимум
            maxPos = i;
        }
// Инкремент счётчика
        i++;
// Накопление значения для расчёта средней мощности
        avgPower += arr[i];
    }
// Вычисление средней мощности сигнала
    avgPower /= 32;
// Возврат максимальной амплитуды
    return max;
}

// Реализация метода для вычисления отношения сигнал/шум (SNR)
int signal::snr(int power){
// Инициализация счётчиков для обхода массива
    uint8_t i = 0, j = 0;
// Вычисление среднего значения сигнала
    int mean = power / 32;
// Цикл по всем выборкам сигнала
    while (i < 32){
// Накопление квадрата разности между выборкой и средним значением
        j += sq(arr[i] - mean);
// Инкремент счётчика
        i++;
    }
// Вычисление и возврат соотношения (корень из квадрата ошибки к мощности)
    return sqrt(j / mean) / power;
}

#include "uspeech.h"
// Реализация конструктора класса syllable
```

```
349 syllable::syllable(){
350 // Инициализация аккумуляторов для каждой фонемы
351     f = 0;
352     e = 0;
353     o = 0;
354     s = 0;
355     h = 0;
356     v = 0;
357 // Инициализация длины слога и временных аккумуляторов фонем
358     length = 0;
359     cf = 0;
360     ce = 0;
361     co = 0;
362     cs = 0;
363     ch = 0;
364     cv = 0;
365 // Инициализация модальности каждой фонемы
366     modalityf = 0;
367     modalitye = 0;
368     modalityo = 0;
369     modalitys = 0;
370     modalityh = 0;
371     modalityv = 0;
372     // Установка флага ожидания пробела в исходное состояние
373     expectSp = 1;
374     // Инициализация счётчика взрывных звуков
375     plosiveCount = 0;
376 }
377
378 // Реализация метода для сброса накопленных значений слога
379 void syllable::reset(){
380 // Сброс аккумуляторов для фонем
381     f = 0;
382     e = 0;
383     o = 0;
384     s = 0;
385     h = 0;
386     v = 0;
387 // Сброс длины слога и временных аккумуляторов
388     length = 0;
389     cf = 0;
390     ce = 0;
391     co = 0;
392     cs = 0;
393     ch = 0;
394     cv = 0;
395 // Сброс модальности для каждой фонемы
396     modalityf = 0;
397     modalitye = 0;
398     modalityo = 0;
399     modalitys = 0;
400     modalityh = 0;
401     modalityv = 0;
402 // Сброс флага ожидания пробела в исходное состояние
403     expectSp = 1;
404 // Сброс счётчика взрывных звуков
405     plosiveCount = 0;
406 }
407
408 //Метод для классификации (фонемы) с обновлением счетчиков
409 void syllable::classify(char c){
410     // Увеличение длины слога (количество обработанных символов)
411     length++;
```

```
412 // Проверка флага ожидания пробела
413     if (expectSp == 0){
414 // Проверка, является ли текущий символ не пробелом
415         if (c != ' '){
416 // Сброс флага ожидания пробела (объединение символов в один слог)
417             expectSp = 1;
418 // Проверка, попадает ли интервал между символами в допустимый порог для
    взрывных звуков
419             if ((millis() - lastTime) < MAX_PLOSIVETIME){
420 // Увеличение счётчика взрывных звуков
421                 plosiveCount++;
422             }
423         }
424     }
425 // Обработка символа в конструкции switch
426     switch (c) {
427 // Обработка символа 'f'
428         case 'f':
429 // Увеличение основного счётчика для 'f'
430             f++;
431 // Увеличение временного счётчика для 'f'
432             cf++;
433 // Выход из оператора switch для 'f'
434             break;
435 // Обработка символа 'e'
436         case 'e':
437 // Увеличение счётчика для 'e'
438             e++;
439 // Увеличение временного счётчика для 'e'
440             ce++;
441             break;
442 // Обработка символа 'o'
443         case 'o':
444 // Увеличение счётчика для 'o'
445             o++;
446 // Увеличение временного счётчика для 'o'
447             co++;
448             break;
449 // Обработка символа 'v'
450         case 'v':
451 // Увеличение счётчика для 'v'
452             v++;
453 // Увеличение временного счётчика для 'v'
454             cv++;
455             break;
456 // Обработка символа 'h'
457         case 'h':
458 // Увеличение счётчика для 'h'
459             h++;
460 // Увеличение временного счётчика для 'h'
461             ch++;
462             break;
463 // Обработка символа 's'
464         case 's':
465 // Увеличение счётчика для 's'
466             s++;
467 // Увеличение временного счётчика для 's'
468             cs++;
469             break;
470 // Обработка символа пробела
471         case ' ':
472 // Проверка активности флага ожидания пробела
473             if (expectSp != 0){
```

```
// Сброс флага ожидания пробела для разделения слогов
                expectSp = 0;
// Фиксация времени, используемого для расчёта интервала между слогами
                lastTime = millis();
            }
            break;
// Обработка любых остальных символов
        default:
            break;
    }
// Периодический анализ для определения пиков и модальности
    if ((length & PROCESS_SKEWNESS_TIME) == 0){
// Анализ временного счётчика для 'f'
        if ((cf > prevf) & (prevf < PROCESS_SKEWNESS_TIME)){
// Обновление предыдущего значения для 'f'
            prevf = cf;
// Фиксация длины слога при достижении пика 'f'
            maxf = length;
// Установка текущего пика и увеличение модальности для 'f'
            if (currPeak != 'f'){
                currPeak = 'f';
                modalityf++;
            }
        }
// Анализ временного счётчика для 'e'
        if ((ce > preve) & (preve < PROCESS_SKEWNESS_TIME)){
            preve = ce;
            maxe = length;
            if (currPeak != 'e'){
                currPeak = 'e';
                modalitye++;
            }
        }
// Анализ временного счётчика для 'o'
        if ((co > prevo) & (prevo < PROCESS_SKEWNESS_TIME)){
            prevo = co;
            maxo = length;
            if (currPeak != 'o'){
                currPeak = 'o';
                modalityo++;
            }
        }
// Анализ временного счётчика для 's'
        if ((cs > prevs) & (prevs < PROCESS_SKEWNESS_TIME)){
            prevs = cs;
            maxs = length;
            if (currPeak != 's'){
                currPeak = 's';
                modalitys++;
            }
        }
// Анализ временного счётчика для 'h'
        if ((ch > prevh) & (prevh < PROCESS_SKEWNESS_TIME)){
            prevh = ch;
            maxh = length;
            if (currPeak != 'h'){
                currPeak = 'h';
                modalityh++;
            }
        }
// Анализ временного счётчика для 'v' с порогом 15
        if ((cv > prevv) & (prevv < 15)){
            prevv = cv;
```

```
            maxv = length;
            if (currPeak != 'v'){
                currPeak = 'v';
                modalityv++;
            }
        }
// Сброс временных счётчиков для нового интервала анализа
        cf = 0;
        ce = 0;
        co = 0;
        cs = 0;
        ch = 0;
        cv = 0;
    }
}

// Реализация метода для возврата указателя на массив с накопленными данными
int* syllable::tointptr(){
    // Объявление статического массива для хранения накопленных значений
    static int matrix[20];
    // Запись значения аккумулятора 'f' в массив
    matrix[0] = f;
    // Запись значения аккумулятора 'e' в массив
    matrix[1] = e;
    // Запись значения аккумулятора 'o' в массив
    matrix[2] = o;
    // Запись значения аккумулятора 'v' в массив
    matrix[3] = v;
    // Запись значения аккумулятора 's' в массив
    matrix[4] = s;
    // Запись значения аккумулятора 'h' в массив
    matrix[5] = h;
    // Запись значения модальности для 'f' в массив
    matrix[6] = modalityf;
    // Запись значения модальности для 'e' в массив
    matrix[7] = modalitye;
    // Запись значения модальности для 'o' в массив
    matrix[8] = modalityo;
    // Запись значения модальности для 'v' в массив
    matrix[9] = modalityv;
    // Запись значения модальности для 's' в массив
    matrix[10] = modalitys;
    // Запись значения модальности для 'h' в массив
    matrix[11] = modalityh;
    // Запись значения пика для 'f' в массив
    matrix[12] = maxf;
    // Запись значения пика для 'e' в массив
    matrix[13] = maxe;
    // Запись значения пика для 'o' в массив
    matrix[14] = maxo;
    // Запись значения пика для 'v' в массив
    matrix[15] = maxv;
    // Запись значения пика для 's' в массив
    matrix[16] = maxs;
    // Запись значения пика для 'h' в массив
    matrix[17] = maxh;
    // Запись значения длины слога в массив
    matrix[18] = length;
    // Запись значения счётчика взрывных звуков в массив
    matrix[19] = plosiveCount;
    // Возврат указателя на заполненный массив
    return matrix;
```

```
599 }
600
601 #include "uspeech.h"
602
603 //Реализация класса statCollector для накопления статистических данных
604 statCollector::statCollector(){
605 // Инициализация переменных для количества выборок, среднего значения и
    моментов нулевыми значениями
606     n = 0;
607     mean = 0;
608     M2 = 0;
609     M3 = 0;
610     M4 = 0;
611 }
612
613 // Реализация метода для возврата среднего значения
614 int statCollector::_mean(){
615     // Возврат рассчитанного среднего значения
616     return mean;
617 }
618
619 // Реализация метода для получения стандартного отклонения
620 int statCollector::stdev(){
621     // Возврат значения второго центрального момента
622     return M2;
623 }
624
625 //Метод для вычисления куртоза - измерение остроты распределения
626 int statCollector::kurtosis(){
627 // Вычисление куртоза по формуле (нормированный 4-й момент минус 3)
628     int kurtosis = (n * M4) / (M2 * M2) - 3;
629 // Возврат вычисленного значения куртоза
630     return kurtosis;
631 }
632
633 // Реализация метода для вычисления асимметрии распределения
634 int statCollector::skew(){
635 // Вычисление асимметрии по формуле (нормированный 3-й момент минус 3)
636     int kurtosis = (n * M3) / (M2 * M2 * M2) - 3;
637 // Возврат полученного значения асимметрии
638     return kurtosis;
639 }
640
641 // Реализация метода для накопления статистических данных с новым
    значением x
642 void statCollector::collect(int x) {
643 // Сохранение предыдущего количества выборок
644     int n1 = n;
645 // Увеличение счётчика выборок
646     n = n + 1;
647 // Вычисление разницы между новым значением и текущим средним
648     int delta = x - mean;
649 // Вычисление поправочного коэффициента для среднего
650     int delta_n = delta / n;
651 // Вычисление квадрата поправочного коэффициента
652     int delta_n2 = delta_n * delta_n;
653 // Вычисление вспомогательного терма для обновления статистических
    моментов
654     int term1 = delta * delta_n * n1;
655 // Обновление среднего значения
656     mean = mean + delta_n;
657 // Обновление четвёртого момента распределения
```

```
658     M4 = M4 + term1 * delta_n2 * (n * n - 3 * n + 3) + 6 * delta_n2 * M2 -
    4 * delta_n * M3;
659 // Обновление третьего момента (асимметрия)
660     M3 = M3 + term1 * delta_n * (n - 2) - 3 * delta_n * M2;
661 // Обновление второго момента (сумма квадратов отклонений)
662     M2 = M2 + term1;
663 }
664
665 #include <Wire.h>
666 // Подключение библиотеки для работы с интерфейсом I²C
667
668 #include <LiquidCrystal_I2C.h>
669 // Подключение библиотеки для управления LCD-дисплеем по шине I²C
670
671 #include <uspeech.h>
672 // Подключение библиотеки uspeech для распознавания речи
673
674 #include <string.h>
675 // Подключение стандартной библиотеки C для работы со строками (strcmp,
    strlen)
676
677 #define ledGreen   7
678 // Определение номера пина 7 для зелёного светодиода
679
680 #define ledOrange  6
681 // Определение номера пина 6 для оранжевого светодиода
682
683 #define ledWhite   5
684 // Определение номера пина 5 для белого светодиода
685
686 #define buzzerPin  2
687 // Определение номера пина 2 для подключения пьезодинамика (буззера)
688
689 #define enablePin  A1
690 // Определение номера аналогового пина A1 для переключателя (замыкание на
    GND = включено)
691
692 #define MIN3(a,b,c) ((a)<(b)?((a)<(c)?(a):(c)):((b)<(c)?(b):(c)))
693 // Макрос для вычисления минимального значения из трёх аргументов
694
695 signal voice(A0);
696 // Создание объекта voice класса signal, подключённого к аналоговому пину
    A0 для сбора аудиосэмплов
697
698 LiquidCrystal_I2C lcd(0x27, 16, 2);
699 // Создание объекта lcd для работы с I²C LCD-дисплеем по адресу 0x27 (16×2
    символа)
700
701 const int  BUFFER_MAX_PHONEMES = 32;
702 // Константа: максимальный размер буфера для накопления фонем
703
704 char       inputString[BUFFER_MAX_PHONEMES];
705 // Массив для накопления входных фонем в виде строки
706
707 byte       index = 0;
708 // Текущий индекс в буфере inputString
709
710 const int  DICT_MAX_ELEMNTS = 3;
711 // Константа: количество элементов (паттернов) в словаре
712
713 char dict[DICT_MAX_ELEMNTS][BUFFER_MAX_PHONEMES] = {
714   "vvvoeeeeeeeofff",   // Паттерн для команды "green"
715   "hhhhhvoovvvvf",     // Паттерн для команды "orange"
```

```
716   "boooooofffffff"    // Паттерн для команды "white"
717 };
718 // Двумерный массив строк с паттернами фонем для распознавания трёх команд
719
720 int    LOWEST_COST_MAX_THREASHHOLD = 20;
721 // Пороговое значение для допустимой стоимости Левенштейна
722
723 unsigned long lastCommandTime      = 0;
724 // Время (в миллисекундах) последнего распознанного и обработанного слова
725
726 bool         idleDisplayed        = false;
727 // Флаг, указывающий, выводилось ли сообщение "ожидание" на дисплей
728
729 bool         voiceEnabled         = false;
730 // Флаг, указывающий, включено ли распознавание речи (состояние
    переключателя)
731
732 //Параметр ms задаёт длительность звукового сигнала в миллисекундах (по
    умолчанию 100 мс).
733 void beep(int ms = 100) {
734   digitalWrite(buzzerPin, LOW);
735   delay(ms);
736   digitalWrite(buzzerPin, HIGH);
737 }
738
739 //Вычисляет длину строки до первого пробела или до символа конца строки
740 int strLength(const char *s) {
741   int i = 0;
742   while (s[i] && s[i] != ' ')
743     ++i;
744   return i;
745 }
746
747 //Вычисляет расстояние Левенштейна между строками s1 и s2.
748 unsigned int levenshtein(char *s1, char *s2) {
749   unsigned int s1len = strlen(s1), s2len = strlen(s2);
750   unsigned int column[s1len + 1];
751 // Инициализация первого столбца матрицы расстояний
752   for (unsigned int y = 1; y <= s1len; y++)
753     column[y] = y;
754 // Основной цикл по символам второй строки
755   for (unsigned int x = 1; x <= s2len; x++) {
756     column[0] = x;
757     unsigned int lastdiag = x - 1;
758     for (unsigned int y = 1; y <= s1len; y++) {
759       unsigned int olddiag = column[y];
760       // Вычисляем стоимость удаления, вставки и замены
761       column[y] = MIN3(
762         column[y] + 1,
763         column[y - 1] + 1,
764         lastdiag + (s1[y - 1] == s2[x - 1] ? 0 : 1)
765       );
766       lastdiag = olddiag;
767     }
768   }
769   return column[s1len];
770 }
771
772 /**
773  * Функция guessWord
774  * По входной строке target вычисляет стоимость Левенштейна до каждого
    паттерна
```

```
775  * и возвращает указатель на наиболее близкий или пустую строку, если ни
     один
776  * паттерн не уложился в порог.
777  */
778 char* guessWord(char* target) {
779   static unsigned int cost[DICT_MAX_ELEMNTS];
780 // Считаем стоимости для каждого паттерна
781   for (int j = 0; j < DICT_MAX_ELEMNTS; j++)
782     cost[j] = levenshtein(dict[j], target);
783 // Ищем лучший (минимальный) cost
784   int best = -1, bestCost = LOWEST_COST_MAX_THREASHHOLD;
785   for (int j = 0; j < DICT_MAX_ELEMNTS; j++) {
786     if (cost[j] < bestCost) {
787       bestCost = cost[j];
788       best = j;
789     }
790   }
791 // Возвращаем либо указатель на строку словаря, либо пустую строку
792   return (best >= 0 ? dict[best] : (char*)"");
793 }
794
795 /**
796  * Функция parseCommand
797  * Обрабатывает распознанное слово: включает нужный светодиод,
798  * выводит текст на дисплей и сбрасывает флаг ожидания.
799  */
800 void parseCommand(char* str) {
801   char *g = guessWord(str);
802   if (!g[0]) return;  // Если слово не распознано — выходим
803
804 // Очистка дисплея и вывод метки
805   lcd.clear();
806   lcd.setCursor(0,0);
807   lcd.print("Word:");
808   lcd.setCursor(0,1);
809
810 // Сначала выключаем все светодиоды
811   digitalWrite(ledGreen,  LOW);
812   digitalWrite(ledOrange, LOW);
813   digitalWrite(ledWhite,  LOW);
814
815 // Сравнение распознанного слова с паттернами через strcmp
816   if      (strcmp(g, dict[0]) == 0) {
817     lcd.print("green");
818     digitalWrite(ledGreen,  HIGH);
819   }
820   else if (strcmp(g, dict[1]) == 0) {
821     lcd.print("orange");
822     digitalWrite(ledOrange, HIGH);
823   }
824   else if (strcmp(g, dict[2]) == 0) {
825     lcd.print("white");
826     digitalWrite(ledWhite,  HIGH);
827   }
828
829 // Обновляем время последней команды и сбрасываем флаг idle
830   lastCommandTime = millis();
831   idleDisplayed   = false;
832 }
833
834 void setup() {
835   Serial.begin(9600);
836   // Инициализация дисплея
```

```
  lcd.init();
  lcd.backlight();
  lcd.clear();

// Настройка пинов на выход
  pinMode(ledGreen,  OUTPUT);
  pinMode(ledOrange, OUTPUT);
  pinMode(ledWhite,  OUTPUT);
  pinMode(buzzerPin, OUTPUT);
  pinMode(enablePin, INPUT_PULLUP);
  digitalWrite(buzzerPin, HIGH); // Выключаем буззер по умолчанию

// Стартовая фаза с индикатором загрузки и калибровкой
  lcd.print("Loading...");
  unsigned long t0 = millis();
  while (millis() - t0 < 5000) {
    unsigned long dt = millis() - t0;
    // Пищим первые 1 секунду
    if (dt < 1000) digitalWrite(buzzerPin, LOW);
    else           digitalWrite(buzzerPin, HIGH);

// После первой секунды выводим сообщение о калибровке
    if (dt >= 1000 && dt < 1250) {
      lcd.clear();
      lcd.print("Calibrating...");
      static bool calibStarted = false;
      if (!calibStarted) {
        calibStarted = true;
        // Настройка параметров распознавания перед калибровкой
        voice.f_enabled = true;
        voice.minVolume  = 1500;
        voice.fconstant  = 500;
        voice.econstant  = 2;
        voice.aconstant  = 4;
        voice.vconstant  = 6;
        voice.shconstant = 10;
        voice.calibrate();
      }
    }

// Мигаем всеми светодиодами в такт
    bool st = ((millis() / 250) & 1);
    digitalWrite(ledGreen,  st);
    digitalWrite(ledOrange, st);
    digitalWrite(ledWhite,  st);
    delay(50);
  }

// После загрузки — выключаем все индикаторы
  digitalWrite(buzzerPin, HIGH);
  digitalWrite(ledGreen,  LOW);
  digitalWrite(ledOrange, LOW);
  digitalWrite(ledWhite,  LOW);

// Читаем состояние переключателя и выводим финальное сообщение
  voiceEnabled = (digitalRead(enablePin) == LOW);
  lcd.clear();
  if (voiceEnabled) lcd.print("Say a color...");
  else              lcd.print("Voice OFF");

  lastCommandTime = millis();
  idleDisplayed   = false;
}
```

```
900 void loop() {
901 // Отслеживаем смену положения переключателя
902   bool en = (digitalRead(enablePin) == LOW);
903   if (en != voiceEnabled) {
904     voiceEnabled = en;
905     // Сбрасываем буфер ввода
906     index = 0;
907     inputString[0] = '\0';
908     // Обновляем экран и при необходимости калибруем микрофон
909     lcd.clear();
910     if (voiceEnabled) {
911       lcd.print("Calibrating...");
912       voice.calibrate();
913       lcd.clear(); lcd.print("Say a color...");
914     } else {
915       lcd.print("Voice OFF");
916       // Выключаем все светодиоды
917       digitalWrite(ledGreen,  LOW);
918       digitalWrite(ledOrange, LOW);
919       digitalWrite(ledWhite,  LOW);
920     }
921     lastCommandTime = millis();
922     idleDisplayed   = false;
923   }
924
925 // Если голосовое управление включено — собираем и обрабатываем фонемы
926   if (voiceEnabled) {
927     voice.sample();
928     char p = voice.getPhoneme();
929 // Если пришёл пробел или буфер полон — обрабатываем накопленную строку
930     if (p == ' ' || index >= BUFFER_MAX_PHONEMES) {
931       if (strLength(inputString) > 0) {
932         parseCommand(inputString);
933         index = 0;
934         inputString[0] = '\0';
935       }
936     } else {
937 // Добавляем фонему в буфер
938       inputString[index++] = p;
939       inputString[index]   = '\0';
940     }
941
942 // Если нет команд больше 5 секунд — показываем приглашение к вводу
943     if (millis() - lastCommandTime > 5000 && !idleDisplayed) {
944       digitalWrite(ledGreen,  LOW);
945       digitalWrite(ledOrange, LOW);
946       digitalWrite(ledWhite,  LOW);
947       lcd.clear();
948       lcd.print("Say a color...");
949       idleDisplayed = true;
950     }
951   }
952 }
```