УДК 004.415.53

ИНТЕГРАЦИЯ СИСТЕМЫ МОНИТОРИНГА И ОЦЕНКИ КАЧЕСТВА ВОЗДУХА В ПОМЕЩЕНИИ С ПОМОЩЬЮ ARDUINO

Сафина Г.Ф.

к. ф.-м. н, доцент,

ФГБОУ ВО «Уфимский университет науки и технологий», Нефтекамский филиал,

Нефтекамск, Россия

Бадрисламов Д.И.

студент,

ФГБОУ ВО «Уфимский университет науки и технологий», Нефтекамский филиал,

Нефтекамск, Россия

Аннотация

В статье рассмотрены основные принципы разработки и интеграции системы мониторинга качества воздуха на базе платформы Arduino. Проведён анализ архитектурных решений, используемых датчиков и методов передачи данных. Показаны преимущества применения микроконтроллера Arduino Uno в сочетании с датчиками MQ-135, DHT22 и PMS5003 для измерения концентрации CO2, летучих органических соединений, температуры, влажности и содержания взвешенных частиц. Представлены примеры кода, демонстрирующие калибровку датчиков, обработку данных и интеграцию с облачными сервисами.

Ключевые слова: Arduino, мониторинг качества воздуха, IoT, датчики газа, микроконтроллер, облачные технологии, экологический мониторинг.

INTEGRATION OF AN INDOOR AIR QUALITY MONITORING AND ASSESSMENT SYSTEM WITH ARDUINO

Safina G.F.
PhD, Associate Professor,
Neftekamsk branch of the Ufa University of Science and Technology,
Neftekamsk, Russia
Badrislamov D.I.
student,
Neftekamsk branch of the Ufa University of Science and Technology,
Neftekamsk, Russia

Abstract

The article discusses the basic principles of development and integration of an air quality monitoring system based on the Arduino platform. The analysis of architectural solutions, sensors used and data transmission methods is carried out. The advantages of using the Arduino Uno microcontroller in combination with MQ-135, DHT22 and PMS5003 sensors for measuring CO2 concentration, volatile organic compounds, temperature, humidity and suspended particles content are shown. Code examples demonstrating sensor calibration, data processing and integration with cloud services are presented.

Keywords: Arduino, air quality monitoring, IoT, gas sensors, microcontroller, cloud technologies, environmental monitoring.

Платформа Arduino (с англ. Automatic Remote Data Input/Output Network Interface – автоматический удалённый интерфейс ввода/вывода данных) представляет собой открытую аппаратно-программную платформу, созданную для быстрой разработки прототипов электронных устройств. Основное преимущество технологии заключается в том, что она устраняет необходимость глубоких знаний в электронике и низкоуровневого программирования, характерного для традиционных микроконтроллеров [1-6].

В отличие от более сложных решений на базе Raspberry Pi, данный инструмент использует прямолинейную, наглядную и прозрачную модель

взаимодействия с периферийными устройствами, сохраняя при этом гибкость конфигурации и богатую экосистему библиотек.

Arduino предлагает упрощенный подход к созданию систем интернета вещей (IoT), основываясь на идеях модульности и простоты программирования. Он предоставляет стандартный набор функций для работы с аналоговыми и цифровыми сигналами, готовые библиотеки для популярных датчиков, а также встроенные средства отладки и мониторинга [1, 5].

Основные преимущества Arduino можно выделить следующим образом:

– простота: программирование на упрощённом C++ с готовыми библиотеками;

– доступность: низкая стоимость компонентов и широкое распространение;

– модульность: возможность легко расширять функциональность с помощью шилдов (дополнительных плат);

– стабильность: предсказуемое поведение в реальном времени.

Arduino органично интегрируется в традиционную трёхслойную архитектуру IoT-устройств. Слой сбора данных реализуется посредством датчиков и аналого-цифровых преобразователей. Для представления сущностей используются простые C++-классы и структуры, содержащие такие элементы, как тип датчика, единицы измерения, пороговые значения и временные метки.

Несмотря на удобства, Arduino имеет свои недостатки:

– ограниченная вычислительная мощность;

– отсутствие многозадачности на уровне ОС;

– сложности при реализации сложных сетевых протоколов;

– требует большего количества ручного кода при работе с внешними сервисами;

– ограниченный объем памяти.

Однако эти ограничения компенсируются предсказуемостью и надёжностью. В случаях, когда необходима сложная обработка данных, можно использовать внешние вычислительные модули или облачные сервисы.

Одним из главных преимуществ Arduino является простая реализация базовых операций сбора данных. Драйверы датчиков создаются на основе классов, расширяющих функциональность базовых сенсоров:

```
    class AirQualityMonitor {
    public:
        virtual float readValue() = 0;
        virtual bool calibrate() = 0;
    };
    class MQ135Sensor : public AirQualityMonitor {
    private:
        uint8_t pin;
        float baseline;
        public:
        MQ135Sensor(uint8_t sensorPin) : pin(sensorPin),
baseline(0.0) {}
            float readValue() override {
            int sensorValue = analogRead(pin);
            float voltage = sensorValue * (5.0 / 1024.0);
            return convertToPPM(voltage);
        }
          bool calibrate() override {
            // Калибровка в чистом воздухе
            float sum = 0;
            for(int i = 0; i < 100; i++) {
                sum += analogRead(pin);
                delay(100);
            }
            baseline = sum / 100;
            return true;
        }
      private:
        float convertToPPM(float voltage) {
            // Конвертация напряжения в PPM
            return 116.602068 * pow((voltage/5.0), -2.769034);
        }
    };
```

Листинг 1 – Базовый класс для работы с датчиками

Класс AirQualityMonitor предоставляет методы: readValue(), calibrate(). В слое доступа к данным создаются специализированные классы, наследуемые от AirQualityMonitor, что позволяет использовать готовые методы для сбора и калибровки данных, чтобы сосредоточиться на логике приложения, а не на низкоуровневом программировании [1, 3, 5].

Приведем пример реализации бизнес-логики в сервисном слое:

```
class AirQualityService {
private:
    MQ135Sensor co2Sensor;
    DHT22Sensor tempHumiditySensor;
    CloudService* cloudService;
    public:
    AirQualityService(uint8_t co2Pin, uint8_t dhtPin)
        : co2Sensor(co2Pin), tempHumiditySensor(dhtPin) {
        cloudService = new CloudService();
    }
    void initializeSensors() {
        co2Sensor.calibrate();
        tempHumiditySensor.begin();
    }
    MonitoringData readAllSensors() {
        MonitoringData data;
        data.timestamp = millis();
        data.co2 = co2Sensor.readValue();
        data.temperature =
tempHumiditySensor.readTemperature();
        data.humidity = tempHumiditySensor.readHumidity();
        return data;
    }
    bool evaluateAirQuality(const MonitoringData& data) {
        return data.co2 < 1000.0 &&
```

```
                data.temperature >= 18.0 &&
                data.temperature <= 26.0 &&
                data.humidity >= 30.0 &&
                data.humidity <= 60.0;
    }
    void sendToCloud(const MonitoringData& data) {
        if (evaluateAirQuality(data)) {
            cloudService->sendData(data);
        }
    }
};
```

Листинг 2 – Пример реализации бизнес-логики

Класс AirQualityService инкапсулирует бизнес-логику работы с датчиками и демонстрирует взаимодействие с различными типами сенсоров. Все операции работают благодаря реализации базовых классов датчиков. Управление измерениями осуществляется с помощью циклического опроса, обеспечивающего регулярный сбор данных.

Для обеспечения взаимодействия C++-объектов с физическими датчиками в Arduino используется механизм отображения (маппинга) программных сущностей на аппаратные компоненты. Это достигается с помощью классов-обёрток, позволяющих явно указать соответствие между программными интерфейсами и физическими пинами, а также между единицами измерения и типами данных.

Представление структуры данных и работы с датчиком DHT22 приведем в следующем листинге.

```
struct MonitoringData {
    unsigned long timestamp;
    float co2;          // в ppm
    float temperature;  // в °C
    float humidity;     // в %
    float pm25;         // в μg/m³
    int airQualityIndex;
```

```
};
class AirQualityMonitor {
public:
    virtual float readValue() = 0;
    virtual bool calibrate() = 0;
    virtual ~AirQualityMonitor() {}
};
class DHT22Sensor : public AirQualityMonitor {
private:
    uint8_t pin;
    DHT dht;
    float lastTemperature;
    float lastHumidity;
    unsigned long lastReadTime;
    const unsigned long READ_INTERVAL = 2000;
 public:
  DHT22Sensor(uint8_t dhtPin) : pin(dhtPin), dht(dhtPin,
DHT22),
lastTemperature(-999.0f), lastHumidity(-999.0f),
lastReadTime(0) {}
        void begin() {
        dht.begin();
        delay(1000);
    }
       float readValue() override {
        return readHumidity();
    }
      float readTemperature() {
        if (millis() - lastReadTime < READ_INTERVAL) {
       return lastTemperature;
        }
       float temp = dht.readTemperature();
        if (isnan(temp)) {
            return lastTemperature;
```

```
    }
    lastTemperature = temp;
    lastReadTime = millis();
    return temp;
}
    float readHumidity() {
    if (millis() - lastReadTime < READ_INTERVAL) {
    return lastHumidity;
    }
   float hum = dht.readHumidity();
    if (isnan(hum)) {
    return lastHumidity;
    }
          lastHumidity = hum;
    lastReadTime = millis();
    return hum;
}
    bool calibrate() override {
    return true;
}
  MonitoringData getMonitoringData() {
    MonitoringData data;
    data.timestamp = millis();
    data.temperature = readTemperature();
    data.humidity = readHumidity();
    data.co2 = -1.0f;
    data.pm25 = -1.0f;
    data.airQualityIndex = -1;
    return data;
}
  bool isDataValid() {
    float temp = dht.readTemperature();
    float hum = dht.readHumidity();
    return (!isnan(temp) && !isnan(hum));
```

```
    }
};
```

Листинг 4 – Представление структуры данных
и работы с датчиком DHT22

Для расширения функциональности Arduino предоставляет возможность использовать различные коммуникационные модули, такие как Ethernet Shield, WiFi модули и GSM модемы [5].

Данные модули позволяют осуществлять передачу данных по сетям, управлять подключениями и обрабатывать ответы без необходимости ручного управления сетевым стеком и другими низкоуровневыми аспектами взаимодействия с сетью.

Представим далее пример листинга использования облачного сервиса.

```
class CloudService {
private:
WiFiClient wifiClient;
const char* server = "api.thingspeak.com";
const char* apiKey = "YOUR_API_KEY";
public:
CloudService() {
WiFi.begin("SSID", "PASSWORD");
}
bool sendData(const MonitoringData& data) {
if (wifiClient.connect(server, 80)) {
String postData = "field1=" + String(data.co2) +
"&field2=" + String(data.temperature) +
"&field3=" + String(data.humidity) +
"&field4=" + String(data.pm25);
String request = "POST /update HTTP/1.1\r\n" +
"Host: " + String(server) + "\r\n" +
"Connection: close\r\n" +
"Content-Type: application/x-www-form-urlencoded\r\n" +
"Content-Length: " + String(postData.length()) + "\r\n\r\n" +
```

```
postData;
wifiClient.print(request);
delay(1000);
wifiClient.stop();
return true;
}
return false;
}
bool isConnected() {
return WiFi.status() == WL_CONNECTED;
}
};
```

Листинг 4 – Пример использования облачного сервиса

В последнем примере выполняется отправка данных мониторинга в облачный сервис. Метод использует HTTP-запрос с параметрами и автоматически преобразует структуру данных в строку запроса.

Такой подход может применяться для долговременного хранения данных, аналитики и удалённого мониторинга, не поддерживаемых в полной мере средствами локального хранения. Использование облачных сервисов расширяет возможности системы мониторинга при сохранении общей архитектуры устройства.

Использование Arduino позволяет реализовать эффективную и предсказуемую систему мониторинга качества воздуха. Благодаря базовым классам датчиков возможно быстро настроить сбор данных с различных сенсоров, а классы-обёртки обеспечивают прозрачное взаимодействие с аппаратными компонентами и поддержку различных типов измерений.

Arduino является гибким инструментом, сочетающим простоту разработки и богатые возможности по работе с периферийными устройствами, что делает её идеальной платформой для создания систем мониторинга окружающей среды.

Библиографический список:

1. Петин В. В. Практическая энциклопедия Arduino: энциклопедия / В. В. Петин, А. А. Биняковский. – М.: ДМК Пресс, 2020. – 166 с.

2. Толстобров А. П. Архитектура ЭВМ: учебник для вузов / А. П. Толстобров. – М.: Юрайт, 2025. – 162 с.

3. Тюкачев Н. А. C#. Основы программирования: учебное пособие для вузов / Н. А. Тюкачев, В. Г. Хлебостроев. – СПбг: Лань, 2021. – 272 с.

4. Бондарь О. Г. Микроконтроллеры в приборах и аппаратах: учебное пособие / О. Г. Бондарь, Е. О. Брежнева. – Вологда: Инфра-Инженерия, 2025. – 200 с.

5. Петин В. В. 77 проектов для Arduino: учебно-методическое пособи / В. В. Петин. – М.: ДМК Пресс, 2020. – 356 с.

6. Ахметов Р.Р., Сафина Г.Ф. Создание "Светофора" и ее программно-аппаратная реализация на микроконтроллере / В сборнике: Первые шаги в науку третьего тысячелетия. Материалы XIII Всероссийской студенческой научно-практической конференции. – 2017. – С. 827-832.