ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ «ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Учебно-методическое пособие

ИНТЕГРАЦИЯ PYTHON И ARDUINO ПРИ ПОМОЩИ ПОСЛЕДОВАТЕЛЬНОГО ИНТЕРФЕЙСА

Ростов-на-Дону – Таганрог
2025 г.

Составители: Хусаинов Н.Ш., Шкурко А.Н., Сазоненко Д.А.

Интеграция Python и Arduino при помощи последовательного интерфейса: Учебно-методическое пособие – Таганрог: ИКТИБ ЮФУ, 2025.– 28 с.

Arduino имеет возможность выводить отладочную и другую информацию в последовательный поток. Мы могли наблюдать за этими выводами в специальном окне Arduino IDE «Serial Monitor». Однако считывать этот поток можно не только с помощью Serial Monitor, который в основном используется для отладки скетчей, но и с помощью скрипта на удобном и простом языке Python. В таком случае, Arduino может управлять процессами на компьютере, а компьютер — процессами на Arduino. Взаимодействие Python и Arduino открывает множество возможностей. Теперь Arduino может реагировать на нажатия пользователя на компьютере, выполняя определенные действия, а также передавать информацию о своей работе или показаниях датчиков на компьютер. Для реализации этого взаимодействия нам потребуется последовательный порт.

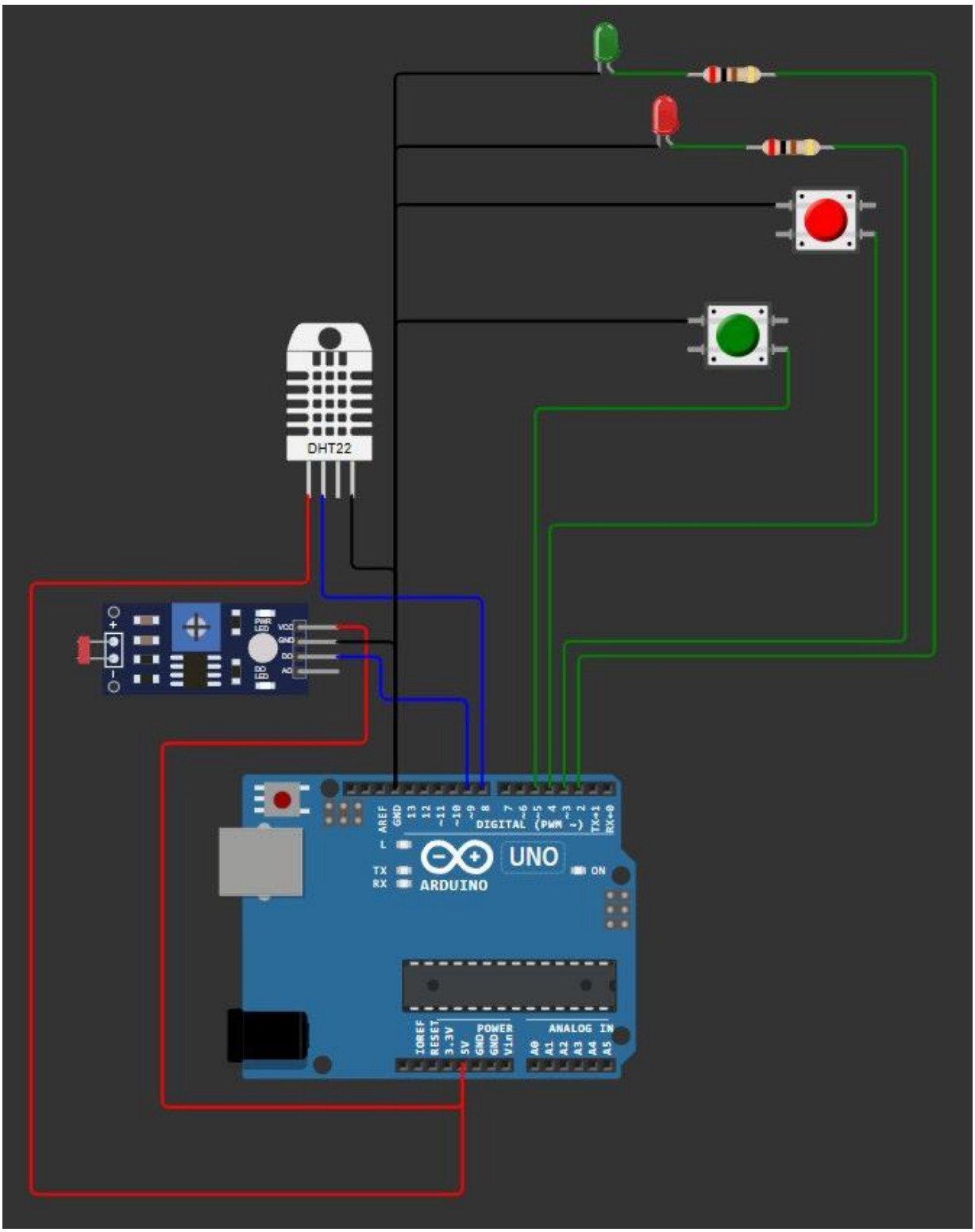
Чтобы понять, как работает взаимодействие между Python и Arduino, давайте создадим приложение на Python, скетч Arduino и схему.

Схема будет состоять из датчиков температуры и света, двух светодиодов, двух резисторов, двух кнопок, соединительных проводов, макетной платы и самого микроконтроллера.

Скетч будет обрабатывать информацию с датчиков и нажатия кнопок, выполнять включение светодиодов. Он должен включать зелёный светодиод, если в помещении темно. Скетч будет обрабатывать нажатия кнопок, увеличивая или уменьшая желаемую пользователем температуру. Как только температура в помещении окажется ниже установленной, то должен включиться красный светодиод.

Приложение на Python будет представлять собой графический интерфейс, где можно будет увидеть текущую температуру в помещении и установленную пользователем температуру. Также в приложении можно будет включать и выключать датчики на Arduino.

Оглавление



Чтобы передать команды от компьютера к Arduino, мы будем использовать строки: «temperature_sensor off», «temperature_sensor on», «light_sensor off», «light_sensor on». Эти строки служат командами для выключения и включения датчиков температуры и света соответственно. Использование строк в качестве команд делает обмен данными более наглядным и понятным. После нажатия соответствующих кнопок в приложении, компьютер будет отправлять эти строки на Arduino. В свою очередь, Arduino должна принять эти команды, определить их и выполнить необходимые действия.

Arduino также должна передавать данные на компьютер. Для этого мы используем строки «<A> needed_temperature X» и «<A> current_temperature Y». Первая строка активируется, когда пользователь меняет желаемую температуру с помощью кнопок, а

вторая — когда датчик температуры выдаёт показания. Дополнительные символы в строках с Arduino служат для проверки корректности команд. Это упрощает понимание команд программистом, а также их проверку и разделение компьютером. Например, компьютер принимает строку вида «<A> needed_temperature 1 <A> needed_temperature2 <A> needed_temperature 3 <A> current_temperature 23.40». Её легко разобрать на составляющие: мы видим четыре команды, три из которых информируют об изменении желаемой пользователем температуры, а четвёртая — о показаниях датчика.

1. Создание скетча Arduino

1.1 Глобальные переменные, константы и библиотеки

Создадим скетч Arduino.

Подключим необходимые библиотеки: OneButton и DHT, отвечающие за удобное использование кнопок и подключение датчика температуры соответственно.

```
     // Библиотека для удобной работы с кнопками

#include <OneButton.h>
     // Библиотека для работы с датчиком температуры

#include <DHT.h>
```

Для удобной работы определим пины, с которыми будем взаимодействовать. Нам понадобятся пины для включения двух светодиодов, получения данных двух кнопок, датчиков света и температуры. Также необходимо определить тип датчика температуры. В нашем случае им будет DHT22.

```
#define PINS_LED_LIGHT 2
#define PINS_LED_BOILER 3
#define PINS_BUTTON_INCTEMP 4
#define PINS_BUTTON_DECTEMP 5
#define DHTPIN 8
#define PINS_LIGHT_SENSOR 9
#define DHTTYPE DHT22
```

Теперь создадим экземпляры кнопок и датчика температуры.

```
     // Кнопка для увеличения необходимой температуры в помещении

OneButton incTemperatureButton = OneButton(PINS_BUTTON_INCTEMP, true,
true);


     // Кнопка для уменьшения необходимой температуры в помещении

OneButton decTemperatureButton = OneButton(PINS_BUTTON_DECTEMP, true,
true);


     // Датчик температуры DHT22

DHT temperatureSensor(DHTPIN, DHTTYPE);
```

Определим необходимые переменные. Для нашего проекта это будут переменные, отвечающие за состояние обоих датчиков, значения температуры и таймеров.

```
     // Таймер для датчика света
```

```
unsigned long lightCheckTimer = 100;

      // Таймер для датчика температуры
unsigned long temperatureCheckTimer = 2000;


      // Текущая температура в помещении
float currentTemperature = 0;

      // Необходимая для пользователя температура
int neededTemperature = 0;
      // Включен ли сейчас датчик света
bool isLightSensorOn = true;
      // Включен ли сейчас датчик температуры
bool isTemperatureSensorOn = true;
```

1.2 Кнопки

Создадим функции для работы наших кнопок. Для кнопки увеличения температуры функция должна увеличивать значение переменной, отвечающей за значение желаемой пользователем температуры. Напишем следующую функцию.

```
void IncTemperature() {
      // Увеличиваем значение необходимой температуры
  neededTemperature++;
      // Обновляем значение необходимой температуры на компьютере
  Serial.print("<A> needed_temperature " + String(neededTemperature) +
" ");
}
```

Для кнопки уменьшения температуры функция будет похожей, но в ней желаемая пользователем температура должна уменьшаться.

```
void DecTemperature() {
      // Уменьшаем значение необходимой температуры
```

```
  neededTemperature--;

      // Обновляем значение необходимой температуры на компьютере

  Serial.print("<A> needed_temperature " + String(neededTemperature) +
" ");

}
```

Что означают строчки обновления температуры на компьютере? В них описана основная идея взаимодействия между Python и Arduino. Микроконтроллер записывает в Serial некоторую информацию в виде строки (в данном случае — температуру). Далее скрипт на Python считывает эту информацию из потока и обрабатывает её. Благодаря этому мы можем вывести текущую температуру окружающей среды на монитор компьютера.

1.3 Начальные настройки

Напишем функцию setup, которая выполнит базовые настройки Arduino. В ней мы должны открыть Serial и установить скорость обмена данными. Также здесь включается датчик температуры, обозначаются функции, вызываемые нажатиями кнопок.

```
void setup() {
      // Начинаем обмен данными со скоростью 9600 Кбит/с

  Serial.begin(9600);
      // Включаем процессы датчика температуры

  temperatureSensor.begin();

      // Устанавливаем режим пина 2 на вывод

  pinMode(PINS_LED_LIGHT, OUTPUT);

      // Устанавливаем режим пина 3 на вывод

  pinMode(PINS_LED_BOILER, OUTPUT);

      // Присоединяем событие нажатия кнопки увеличения желаемой
температуры к функции увеличения желаемой температуры

  incTemperatureButton.attachClick(IncTemperature);

      // Присоединяем событие нажатия кнопки уменьшения желаемой
температуры к функции уменьшения желаемой температуры

  decTemperatureButton.attachClick(DecTemperature);

}
```

1.4 Основной цикл программы

Осталось только создать функцию loop, которая будет управлять всеми процессами на Arduino. В этой функции предстоит выполнить множество важных задач: обработать нажатия на обе кнопки, обновлять показания датчиков температуры и освещённости, а также принимать решения в зависимости от уровня освещенности и температуры. Кроме того, в этой функции необходимо считывать данные, отправляемые компьютером.

```
void loop() {
     // Производим один тик для каждой кнопки, то есть проверяем их
состояние и относительно него OneButton выполнит действия

  incTemperatureButton.tick();

  decTemperatureButton.tick();


     // Если датчик света включен и с прошлого момента обновления его
показаний прошло не менее 100 мс, то пора получить актуальную
информацию

  if (isLightSensorOn && (millis() - lightCheckTimer >= 100)) {
     // Если освещённость низкая — пора включить свет

    digitalWrite(PINS_LED_LIGHT, digitalRead(PINS_LIGHT_SENSOR) ? HIGH
: LOW);
     // Запоминаем, когда в последний раз обновили показания

    lightCheckTimer = millis();

  }


     // Если с прошлого момента обновления показаний датчика
температуры прошло не менее 2000 мс, то пора обновить его показания,
если он включен

  if (isTemperatureSensorOn && millis() - temperatureCheckTimer >=
2000) {
     // Получаем значение температуры в помещении с датчика DHT22

    currentTemperature = temperatureSensor.readTemperature();
     // Если меньше — включаем «нагреватель», иначе - выключаем

    digitalWrite(PINS_LED_BOILER, (currentTemperature <=
neededTemperature) ? HIGH : LOW);
     // Информируем компьютер о значении текущей температуры

    Serial.print("<A> current_temperature " +
String(currentTemperature));

     // Запоминаем, когда в последний раз обновили показания

    temperatureCheckTimer = millis();

  }
```

```
      // Если есть данные из Serial

  if (Serial.available() > 0) {
      // Считываем строку, отправленную компьютером

    String receivedData = Serial.readString();



      // Если принятая строка — это команда включения датчика
температуры, то включаем
    if (receivedData == "temperature_sensor on") isTemperatureSensorOn
= true;
      // Если принятая строка — это команда выключения датчика
температуры, то выключаем

    else if (receivedData == "temperature_sensor off")
isTemperatureSensorOn = false;



      // Если принятая строка — это команда включения датчика света, то
включаем

    if (receivedData == "light_sensor on") isLightSensorOn = true;
      // Если принятая строка — это команда выключения датчика света,
то выключаем

    else if (receivedData == "light_sensor off") isLightSensorOn =
false;

  }

}
```

В этом фрагменте кода используется функция millis(), которая позволяет узнать текущее время с момента запуска программы. Переменная receivedData представляет собой строку, что делает команды простыми и понятными для программиста. Сравнивая эту строку с заранее заданными командами, Arduino может определить, какое действие необходимо выполнить после получения конкретной команды.

Скетч для Arduino успешно завершён. Теперь микроконтроллер способен самостоятельно определять уровень освещённости и температуру в помещении, включать и выключать светодиоды и реагировать на нажатие кнопок. Следующим шагом будет написание скрипта на Python, который позволит реализовать основную идею проекта — взаимодействие Arduino с компьютером.

2. Создание программы на Python

2.1 Виртуальное окружение Python

Перед тем как приступить к работе с Python, важно установить все необходимые библиотеки. Однако, не рекомендуется устанавливать их в глобальную директорию, особенно если компьютером пользуются несколько человек. Некоторые библиотеки могут конфликтовать друг с другом, и для корректной работы одной из них потребуется удаление другой. Чем больше библиотек будет установлено в глобальную директорию, тем выше риск возникновения проблем. Чтобы избежать подобных ситуаций, мы рекомендуем использовать виртуальное окружение Python. Это изолированная среда, которая позволяет работать с различными версиями языка и отдельными библиотеками, не затрагивая основную директорию.

Давайте **создадим виртуальное окружение Python** в папке нашего проекта, используя следующие команды:

```
mkdir project

cd project

python -m venv env
```

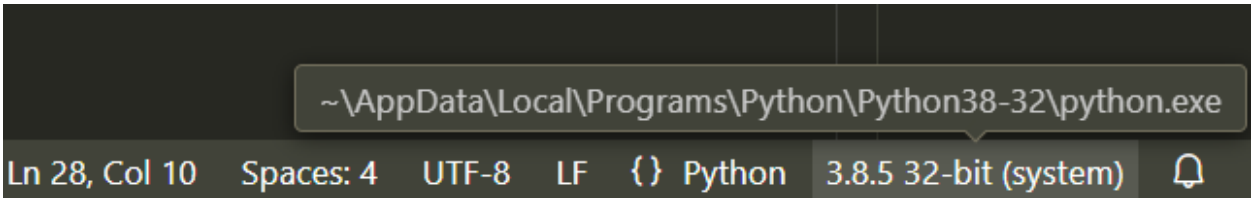
Первые две служат для создания папки с виртуальным окружением и её открытия. Третья команда – это создание самого виртуального окружения с названием «env».

Чтобы активировать окружение, необходимо запустить скрипт activate.bat, расположенный в папке с виртуальным окружением. По завершении работы следует воспользоваться скриптом deactivate.bat для завершения процесса. Предположим, виртуальное окружение было создано в директории «e:\home\python\project». Тогда процесс активации, установки необходимых пакетов и деактивации будет выглядеть следующим образом:

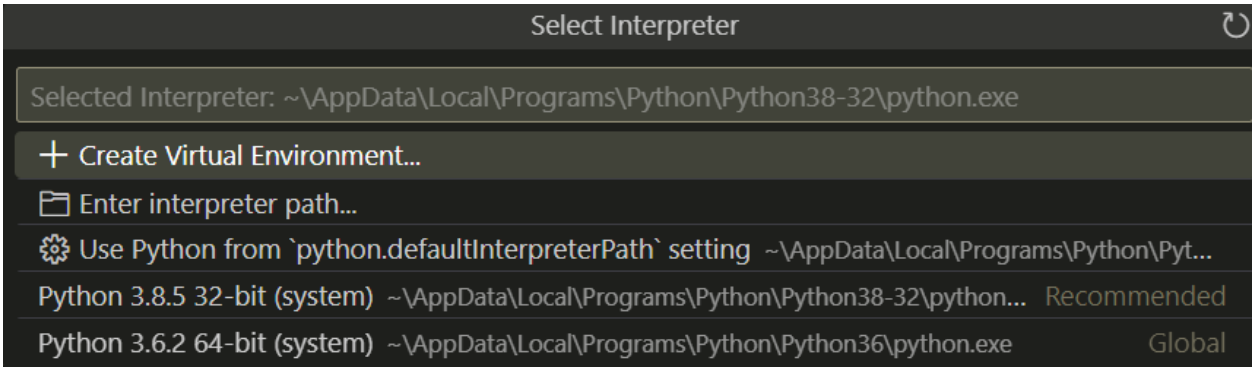
```
e:\home\python\project\env\Scripts\activate.bat

pip install requests

e:\home\python\project\env\Scripts\deactivate.bat
```

Теперь нужно, чтобы Visual Studio Code, которую мы будем использовать при разработке приложения, при запуске кода **использовала именно виртуальную среду**. Для этого откроем VS Code, создадим новый файл Python. В нижнем правом углу экрана будет текущая версия интерпретатора.

Если кликнуть на эту надпись, то появится возможность выбора нужной версии Python. Укажем путь к python.exe в виртуальном окружении env, созданном нами недавно. После этих действий VS Code будет без проблем запускать программы в виртуальном



окружении.



Виртуальное окружение создано, среда разработки настроена, теперь можно переходить к установке в него необходимых библиотек.

2.2 Библиотеки

Для работы с COM-портами в Python необходима библиотека PySerial. Установить PySerial можно с помощью менеджера пакетов Python — pip. Для этого откройте терминал или командную строку и выполните следующую команду:

```
pip install pyserial
```

Для создания графических интерфейсов в Python лучше использовать библиотеку PyQt5. Она предоставляет удобные инструменты для разработки десктопных приложений. Установить PyQt5 также можно с помощью pip. В терминале или командной строке введите следующую команду:

```
pip install PyQt5
```

После выполнения этих команд необходимые библиотеки будут установлены и готовы к использованию.

Перейдём к написанию кода. Давайте напишем программу, которая по нажатию кнопок в ней может включать и выключать датчики на Arduino, принимать информацию с одного из датчиков и выводить её на экран. Для начала нужно прописать несколько import.

```
import sys

import serial

import serial.tools.list_ports

from PyQt5.QtWidgets import *

from PyQt5.QtGui import *

from PyQt5.QtCore import *

from threading import Thread
```

Библиотека sys предоставляет доступ к системным функциям и переменным, что позволяет управлять аргументами командной строки и завершать выполнение программы при необходимости.

Библиотека serial, а также модуль serial.tools.list_ports из нее, необходимы для работы с последовательными портами, что позволяет устанавливать соединение с устройствами и осуществлять обмен данными между ними.

Далее нужны компоненты из библиотеки PyQt5: QtWidgets, QtGui и QtCore. Эти модули обеспечивают создание графического пользовательского интерфейса (GUI), позволяя разрабатывать визуально привлекательные и интерактивные приложения.

Не менее важным является Thread из модуля threading. Он позволяет создать отдельные потоки выполнения. Это даёт возможность сохранять отзывчивость приложения во время выполнения команд с Arduino. Если поток в скрипте не открывать, то приложение будет виснуть во время чтения и обработки информации с Arduino, а также в процессе выполнения посланных микроконтроллером команд.

2.3 Каркас приложения

Создадим наше графическое приложение. Для этого опишем класс, являющийся наследником QMainWindow – он будет отвечать за главное окно нашего приложения.

```
class ControllerApp(QMainWindow):
```

Опишем несколько переменных, необходимых для реализации желаемого функционала. Также создадим сигналы (позволяют объектам сообщать друг другу о конкретных событиях).

```
# Переменная, отвечающая за завершение потока чтения с Arduino
    end_thread = False
# Переменная для хранения сообщения, полученного от Arduino
    message_from_arduino = ""
# Переменная, указывающая, работает ли сейчас датчик света
    is_light_sensor_ON = True
# Переменная, указывающая, работает ли сейчас датчик температуры
    is_temperature_sensor_ON = True
# Сигнал для обновления текущей температуры
    update_current_temperature_signal = pyqtSignal(str)
# Сигнал для обновления необходимой температуры
```

```
    update_needed_temperature_signal = pyqtSignal(str)
```

Чтобы класс можно было создавать в коде, ему нужен конструктор. Для этого опишем функцию __init__ в нашем классе. Сначала она должна инициализировать базовый класс QMainWindow. В ней же будем задавать стандартные значения для всего в приложении.

```
    def __init__(self):

# Инициализация базового класса QMainWindow

        super().__init__()
```

Здесь же настроим главное окно:

```
# Устанавливаем размеры и положение окна: верхний левый угол находится
в координатах (500, 500), а размеры окна составляют 500 пикселей в
ширину и 800 пикселей в высоту.

        self.setGeometry(500,500,500,800)

# Получаем текущую геометрию окна, включая рамку, и сохраняем её в
переменной temp1

        temp1 = self.frameGeometry()

# Перемещаем центр окна в центр доступной области экрана (изменяем
информацию о нем)


temp1.moveCenter(QDesktopWidget().availableGeometry().center())

# Фактически устанавливаем верхний левый угол окна в новое положение,
чтобы окно оказалось в центре экрана

        self.move(temp1.topLeft())

# Задаём заголовок окна

        self.setWindowTitle('Arduino and Python interaction')
```

2.4 Меню для соединения с Arduino

Чтобы человек мог пользоваться нашим приложением, оно должно грамотно подключаться к Arduino. Но указывать статический порт или делать автоматический подбор подходящего порта нельзя из-за возможности подключения к ПК нескольких Arduino. Потому нужно дать пользователю возможность самостоятельно выбрать нужный девайс. Для этого создадим выпадающий список всех доступных портов.

```
# Создаем экземпляр QComboBox
```

```
        self.port_combo = QComboBox()
# Перебираем все доступные последовательные порты с помощью метода
comports() из библиотеки (возвращает все доступные COM-порты)

        for port in serial.tools.list_ports.comports():
# Проверяем, содержит ли имя порта строку "ttyS". К нашей задаче они
не относятся

            if "ttyS" in port.device:

                continue
# Добавляем имя порта в выпадающий список, если он не был пропущен

            self.port_combo.addItem(port.device)
```

Нужно создать кнопки для управления подключением и отключением от портов. К сигналам, поступающим при нажатии на эти кнопки, будут привязаны функции подключения и отключения. Эти функции напишем позже.

```
# Создаем кнопку "Connect" для подключения к порту

        self.connect_button = QPushButton("Connect")

# Подключаем сигнал нажатия кнопки к методу connectSerial

        self.connect_button.clicked.connect(self.connectSerial)



# Создаем кнопку "Disconnect" для отключения от порта

        self.disconnect_button = QPushButton("Disconnect")

# Подключаем сигнал нажатия кнопки к методу disconnectSerial

        self.disconnect_button.clicked.connect(self.disconnectSerial)
```

Теперь у пользователя есть необходимые инструменты в виде кнопок и списка для работы с подключением к Arduino. Выделим их в отдельную панель инструментов. Для этого сначала создадим её,

```
# Создаем панель инструментов с названием 'Tools'

        self.toolbar = self.addToolBar('Tools')

# Устанавливаем фиксированную высоту панели инструментов в 50 пикселей

        self.toolbar.setFixedHeight(50)
```

а потом добавим туда все необходимые виджеты.

```
# Добавляем выпадающий список портов на панель инструментов

        self.toolbar.addWidget(self.port_combo)
# Добавляем кнопку "Connect" на панель инструментов
```

```
        self.toolbar.addWidget(self.connect_button)
# Добавляем кнопку "Disconnect" на панель инструментов

        self.toolbar.addWidget(self.disconnect_button)
```

2.5 Настройка кнопок

Разберёмся с элементами основного функционала приложения. Создадим и настроим кнопки для включения и выключения датчиков света и температуры, по нажатию на которые должны выполнятся соответствующие действия с датчиками.

```
# Создаем кнопку для включения и выключения датчика света

        self.temperature_sensor_button = QPushButton(self)

# Устанавливаем текст кнопки

        self.temperature_sensor_button.setText("TURN <OFF> LIGHT
SENSOR")

# Устанавливаем геометрию кнопки: (x, y, ширина, высота)

        self.temperature_sensor_button.setGeometry(250, 50, 250, 250)

# Подключаем сигнал нажатия кнопки к методу temperatureSensorTurn


self.temperature_sensor_button.clicked.connect(self.temperatureSensorT
urn)



# Создаем кнопку для включения и выключения датчика температуры

        self.light_sensor_button = QPushButton(self)

# Устанавливаем текст кнопки

        self.light_sensor_button.setText("TURN <OFF> TEMPERATURE
SENSOR")

# Устанавливаем геометрию кнопки: (x, y, ширина, высота)

        self.light_sensor_button.setGeometry(0, 50, 250, 250)

# Подключаем сигнал нажатия кнопки к методу lightSensorTurn

        self.light_sensor_button.clicked.connect(self.lightSensorTurn)
```

Также создадим текстовые поля для информации с Arduino. Добавим поле, в котором будет отображаться текущая температура окружающей среды,

```
# Создаем информационное поле для отображения текущей температуры

        self.current_temperature_label = QLabel(self)
```

```
# Устанавливаем геометрию поля: (x, y, ширина, высота)
        self.current_temperature_label.setGeometry(0, 300, 500, 50)
# Устанавливаем начальный текст поля
        self.current_temperature_label.setText("No connection!")
# Устанавливаем выравнивание текста по центру
        self.current_temperature_label.setAlignment(Qt.AlignCenter)
```

и поле, в котором будет отображаться информация о выставленной на микроконтроллере желаемой температуре в помещении.

```
# Создаем информационное поле для отображения выставленной температуры
        self.needed_temperature_label = QLabel(self)
# Устанавливаем геометрию поля: (x, y, ширина, высота)
        self.needed_temperature_label.setGeometry(0, 350, 500, 50)
# Устанавливаем начальный текст поля
        self.needed_temperature_label.setText("No connection!")
# Устанавливаем выравнивание текста по центру
        self.needed_temperature_label.setAlignment(Qt.AlignCenter)
```

Для обновления температуры на соответствующих текстовых полях будем использовать сигналы. Они позволяют исправно взаимодействовать с элементами графического интерфейса из другого потока. Как поток будет выглядеть, мы разберёмся позже.

```
# Подключаем сигналы к соответствующим методам для обновления
температуры

self.update_current_temperature_signal.connect(self.updateCurrentTempe
rature)

self.update_needed_temperature_signal.connect(self.updateNeededTempera
ture)
# Инициализируем переменную для хранения соединения с последовательным
портом
        self.serial_connection = None
```

Теперь осталось лишь отобразить панель инструментов и само главное окно приложения.

```
# Отображаем панель инструментов
        self.toolbar.show()
# Показываем главное окно приложения
        self.show()
```

2.6 Функционал кнопок и отправка данных на Arduino

Конструктор класса основного окна приложения завершён, и можно переходить к написанию других методов класса. Создадим метод для подключения к порту.

```
# Определяем метод для подключения к последовательному порту
    def connectSerial(self):
```

Как только программа подключается к порту, мы должны провести несколько базовых настроек. Основным действием здесь будет установка соединения через создание Serial. Для грамотной работы отрисовки, отзывчивости приложения во время его работы и правильного процесса обмена данных с Arduino откроем новый поток для чтения данных, в котором будет выполняться соответствующая функция.

```
# Устанавливаем флаг завершения потока в значение False
        self.end_thread = False
# Отключаем кнопку подключения, чтобы предотвратить повторные нажатия
        self.connect_button.setEnabled(False)
```

```
# Устанавливаем соединение с последовательным портом, используя выбранный порт и скорость 9600
        self.serial_connection = serial.Serial(self.port_combo.currentText(), 9600, timeout=0.3)
```

```
# Создаем и запускаем новый поток для чтения данных с Arduino
        self.arduino_thread = Thread(target=self.arduinoThreadTick)
        self.arduino_thread.start()
```

Создадим метод для отключения от порта. В этом методе необходимо просто сообщить программе о завершении работы потока, так как процесс закрытия порта будет выполнен позже в потоке, отвечающем за чтение информации с Arduino.

```
    def disconnectSerial(self):
# Завершаем поток обмена информации между компьютером и Arduino
```

```
        self.end_thread = True
```

Мы устанавливаем флаг для завершения потока обмена данными с Arduino. Это действие прерывает активное соединение для передачи данных, и его необходимо выполнить, чтобы впоследствии поток не пытался обратиться к несуществующему соединению. Завершение потока должно происходить и при завершении программы, поэтому мы переопределим функцию завершения программы.

```
    def closeEvent(self, event):

# Устанавливаем флаг завершения потока

        self.end_thread = True
```

Создадим метод для обновления информации о температуре окружающей среды, который по входящей температуре будет устанавливать текст в соответствующее текстовое поле.

```
    def updateCurrentTemperature(self, temperature):

# Обновляем текст метки текущей температуры

        self.current_temperature_label.setText("Current temperature: "
+ temperature)
```

Создадим метод для обновления информации о желаемой пользователем температуре, который по входящей температуре будет устанавливать текст в соответствующее текстовое поле.

```
    def updateNeededTemperature(self, temperature):

# Обновляем текст метки желаемой температуры

        self.needed_temperature_label.setText("Needed temperature: " +
temperature)
```

Теперь перейдём к реализации функционала кнопок включения и выключения датчиков света и температуры. Для начала создадим метод для взаимодействия компьютера с датчиком света.

```
    def lightSensorTurn(self):

# Проверяем, включен ли датчик света

        if (self.is_light_sensor_ON):

# Если включен, отправляем команду на отключение датчика
```

```
            self.serial_connection.write("light_sensor
off".encode('utf-8'))

        else:

# Если выключен, отправляем команду на включение датчика

            self.serial_connection.write("light_sensor
on".encode('utf-8'))



# Обновляем текст кнопки в зависимости от состояния датчика света

        if (self.is_light_sensor_ON):

            self.light_sensor_button.setText("TURN <ON> LIGHT SENSOR")

        else:

            self.light_sensor_button.setText("TURN <OFF> LIGHT
SENSOR")



# Переключаем состояние датчика света

        self.is_light_sensor_ON = not(self.is_light_sensor_ON)
```

Создадим метод для взаимодействия компьютера с датчиком температуры.

```
    def temperatureSensorTurn(self):

# Проверяем, включен ли датчик температуры

        if (self.is_temperature_sensor_ON):

# Если включен, отправляем команду на отключение датчика

            self.serial_connection.write("temperature_sensor
off".encode('utf-8'))

        else:

# Если выключен, отправляем команду на включение датчика

            self.serial_connection.write("temperature_sensor
on".encode('utf-8'))



# Обновляем текст кнопки в зависимости от состояния датчика
температуры
```

```
        if (self.is_temperature_sensor_ON):

            self.temperature_sensor_button.setText("TURN <ON>
TEMPERATURE SENSOR")

        else:

            self.temperature_sensor_button.setText("TURN <OFF>
TEMPERATURE SENSOR")



# Переключаем состояние датчика температуры

        self.is_temperature_sensor_ON =
not(self.is_temperature_sensor_ON)
```

Теперь перейдём к самой важной функции всей программы: методу чтения данных с Arduino. Здесь должно происходить несколько важных действий. Во-первых, завершение потока при необходимости, во-вторых, принятие данных с микроконтроллера, в-третьих, их обработка и выполнение соответствующих действий.

2.7 Получение данных с Arduino

```
    def arduinoThreadTick(self):
```

Этот поток будет запускаться при открытии какого-либо порта. Нам нужно, чтобы он работал непрерывно до момента закрытия программы или отключения от порта.

```
# Запускаем бесконечный цикл для постоянного чтения данных

        while True:

# Проверяем, установлен ли флаг завершения потока

            if self.end_thread == True:

# Закрываем соединение с последовательным портом

                self.serial_connection.close()

                return
```

В самом цикле нужно написать процесс чтения и обработки данных, которые посылает Arduino. Команды заранее заданы, потому будем просто принимать последовательность слов, разбирать их на команды и выполнять самые актуальные.

```
# Читаем строку данных из последовательного порта и декодируем её в
UTF-8

            self.message_from_arduino =
self.serial_connection.readline().decode('utf-8')

# Проверяем, что длина полученного сообщения больше 3 символов. Это
нужно, чтобы избегать лишней обработки пустых строчек. 3 символа
выставлены с расчётом на присутствие во всех командах подстроки «<A>»

            if (len(self.message_from_arduino) > 3):

# Разворачиваем список слов, полученных из сообщения

                message =
list(reversed(self.message_from_arduino.split()))



# Если в сообщении содержится "current_temperature", обновляем текст
на экране с текущей температурой

                if "current_temperature" in self.message_from_arduino:


self.update_current_temperature_signal.emit(message[message.index("cur
rent_temperature") - 1])

# Если в сообщении содержится "needed_temperature", обновляем текст на
экране с необходимой температурой

                if "needed_temperature" in self.message_from_arduino:


self.update_needed_temperature_signal.emit(message[message.index("need
ed_temperature") - 1])
```

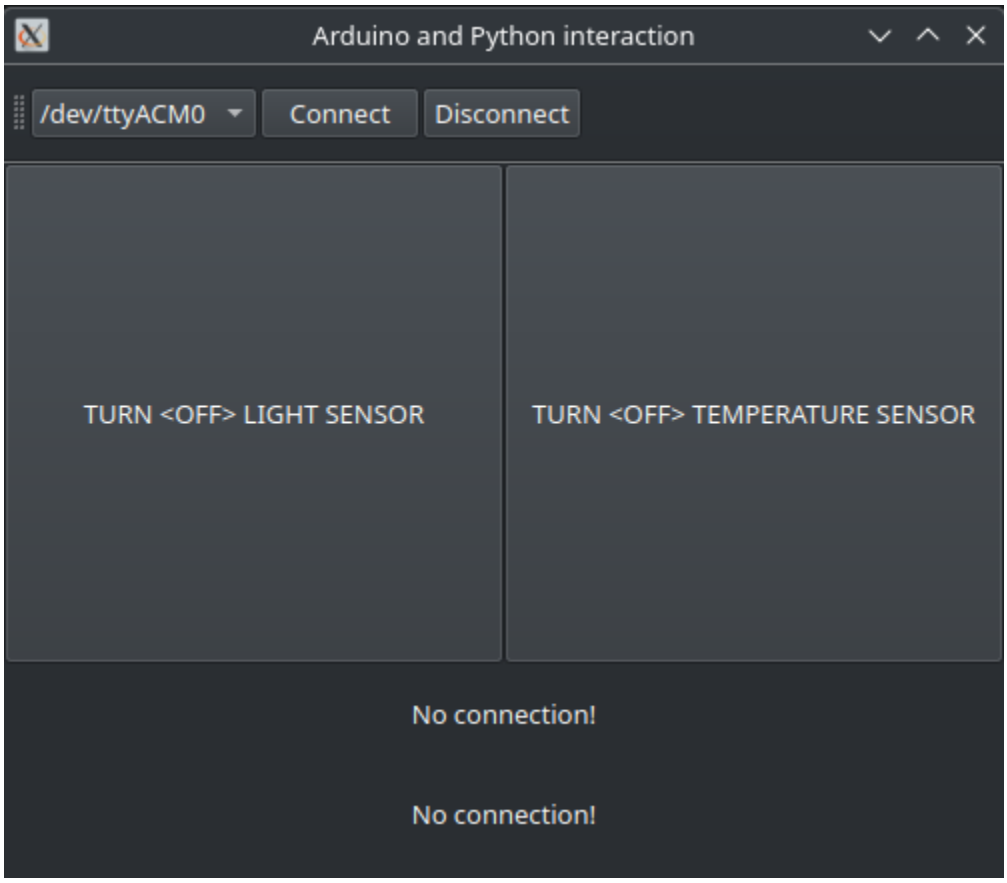
2.8 Главная функция приложения

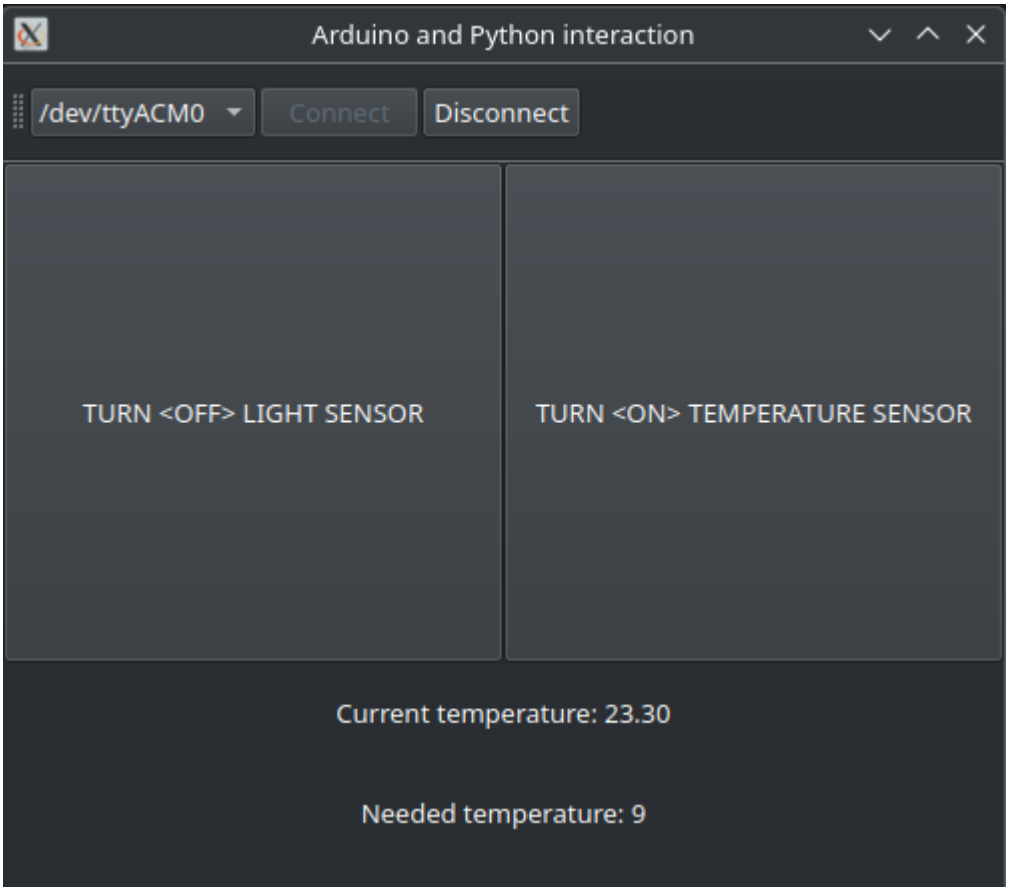
Теперь осталось лишь написать функцию main для работы программы. В ней нам надо создать экземпляр класса нашего приложения и запустить его главный цикл.

```
if __name__ == '__main__':

    # Создаем экземпляр приложения

    app = QApplication(sys.argv)

    # Создаем экземпляр контроллера приложения

    arduino_controller = ControllerApp()
```

```
# Запускаем главный цикл приложения

sys.exit(app.exec_())
```

3. Галерея


Arduino and Python interaction
/dev/ttyACM0
Connect
Disconnect
TURN <OFF> LIGHT SENSOR
TURN <OFF> TEMPERATURE SENSOR
No connection!
No connection!


Arduino and Python interaction
/dev/ttyACM0
Connect
Disconnect
TURN <OFF> LIGHT SENSOR
TURN <ON> TEMPERATURE SENSOR
Current temperature: 23.30
Needed temperature: 9

4. Варианты лабораторных работ для выполнения

Во всех вариантах также обязательная реализация выбора порта для подключения из списка доступных.

Вариант 1. Охрана дома.

Нужно защитить дом от несанкционированного доступа. Для этого использовать датчик движения. Датчик касания позволит включить систему в режим охраны. При этом должен включаться зелёный светодиод. Если срабатывает датчик движения, то должен включиться красный светодиод. Если в приложении в течение минуты не была нажата кнопка отмены тревоги, то должна включиться сирена. В приложении должна выводиться информация о состоянии датчика, светодиодов и тревоги. Также в нём должны находиться кнопки включения/отключения датчика движения и включения/отключения звуковых сигналов.

Датчики (сенсоры) для получения данных	**Устройства управления / отображения**
1) Датчик касания 2) HC-SR501 (PIR датчик)	1) Светодиоды (красный и зеленый) 2) Пьезоизлучатель

Вариант 2. Пожарная сигнализация.

Необходимо реализовать простую пожарную сигнализацию, которая помимо непосредственно сигнализирования о пожаре, будет оповещать о потенциальном возгорании. В случае обнаружения горения датчиком пламени должна немедленно включаться сирена и моргать светодиод. Для отслеживания потенциального горения должен использоваться датчик газов. Для этого датчика должно быть введено 2 пороговых уровня. При превышении меньшего порога должен включаться светодиод и тихая сирена (редкие короткие звуки, например). При превышении большего порога должна включаться громкая сирена и моргать светодиод. Такая же логика должна быть реализована для датчика температуры. В приложении должны выводиться показания всех датчиков (достаточно трёх пунктов) и текущее состояние системы (тревога или нет). Также должны быть кнопки для включения/отключения звуковых и световых сигналов.

Датчики (сенсоры) для получения данных	**Устройства управления / отображения**
1) Датчик пламени (огня) 2) Датчик широкого спектра газов MQ-2 3) DHT22 Temperature-Humidity Sensor	1) Светодиоды 2) Пьезоизлучатель

Вариант 3. Комфортная среда для растений

Для создания комфортной среды для растений, нужно чтобы воздух был чистым, умеренно влажным, теплым, а растения получали достаточно влаги и света. Но за этим всегда нужен контроль. Потому устройство, которое могло бы помочь нам с этим, будет очень полезно. Сделайте устройство, которое будет отслеживать влажность и температуру воздуха. Если влажность меньше 30% или больше 70%, на дисплее должно отображаться соответствующее предупреждение. Также устройство должно отслеживать освещенность для растений. Если свет выключается более чем на 10 секунд, мы должны получать звуковой сигнал об этом. Ну, и чтобы растения всегда были вовремя политы, устройство должно отслеживать влажность почвы. Если влажность подходящая, то

светодиоды не горят. Если влажность умеренная, то загорается один светодиод, а если влажность низкая, то должны включиться оба. Показания всех датчиков должны выводиться в приложении (достаточно трёх пунктов). Должны быть кнопки для включения/отключения звуковых и световых сигналов.

Датчики (сенсоры) для получения данных	**Устройства управления / отображения**
1) DHT22 Temperature-Humidity Sensor (HDC1080)	1) Светодиоды (2 разных цветов)
2) Датчик влажности почвы	2) Пьезоизлучатель
3) Фоторезистор	

Вариант 4. Управление предметами по времени.

Чтобы сделать дом более умным иногда достаточно простой логики управления предметами по времени. Очевидно, это может быть достаточно просто реализовано с использованием МК. Вам нужно в определенное время на улице включать свет и так же выключать. Если все легли спать и забыли поставить дом на сигнализацию, включите её в нужное время. Время включения сигнализации и света должно настраиваться при помощи кнопок. Однако, в разное время года темнеет в разное время, а значит запрограммированное на одно время включение света может в какое-то время прийтись на светлое время суток. Чтобы этого избежать, используйте фоторезистор. Если на улице светло, то свет должен включиться только когда стемнеет. Состояние систем, настройка времени, а также часы должны постоянно отображаться в приложении. Также должны быть кнопки включения/отключения звуковых сигналов и фоторезистора.

Датчики (сенсоры) для получения данных	**Устройства управления / отображения**
1) Часы реального времени DS3231	1) Светодиоды
2) Тактовые кнопки (2-3 шт)	2) Пьезоизлучатель
3) Фоторезистор	

Вариант 5. Защита от затопления дома.

Трубы имеют свойство ржаветь, а стиральная машинка имеет свойство ломаться и протекать. Хорошо бы узнать о подобных проблемах сразу, чтобы минимизировать ущерб. Поэтому для контроля таких ситуаций будем использовать датчик влажности почвы, который сообщит нам, о том, что где-то произошла протечка воды. А если Вы любите купаться в полные воды ванной, но при этом легко забываете вовремя выключить воду или Вас в этот момент могут отвлечь, то будем использовать датчик определения уровня воды. И конечно нужно сразу сигнализировать о произошедшей ситуации (при помощи пьезоизлучателя). Также неплохо бы было включать вентилятор в ванной при превышении заданного порога влажности (роль вентилятора будет исполнять светодиод). Температура и влажность в ванной, а также состояние системы должны отображаться в приложении. Также в нём должны быть кнопки включения/отключения звуковых сигналов и датчика температуры и влажности.

Датчики (сенсоры) для получения данных	**Устройства управления / отображения**
1) Датчик уровня воды (угловой)	1) Светодиоды
2) Датчик влажности почвы	2) Пьезоизлучатель
3) Датчик температуры и влажности DHT11/22	

Вариант 6. Комфортный дом.

Хорошо когда техника заботится о комфорте проживающих там людей. Для этого иногда даже не обязательно делать много. Иногда достаточно, чтобы в доме просто было светло и тепло. Потому вам нужно разработать устройство, которое будет управлять светом и теплом в доме. Один из светодиодов, которые у вас есть, будет отвечать за свет, а другой за нагреватель, установленный в доме. Желаемую температуру в доме можно менять при помощи тактовых кнопок (повысить/понизить). Если температура в доме ниже заданной, то нагреватель должен включаться (один из светодиодов), если выше – выключаться. Если в доме становится темно - должен включаться свет (второй светодиод). Текущая температура в доме, атмосферное давление, влажность и желаемая пользователем температура должны отображаться в приложении. Также в нём должны быть кнопки включения/отключения всех датчиков.

Датчики (сенсоры) для получения данных	**Устройства управления / отображения**
1) Датчик BMP280	1) Светодиоды (2 шт)
2) Датчик освещенности	3) Тактовые кнопки
3) Датчик HDC1080 (DHT22)	

Вариант 7. Мониторинг здоровья пациентов.

Медицинский персонал в больницах часто использует специальные устройства, которые отслеживают основные показатели здоровья пациентов. Часто эти устройства дорогие, но позволяют получать показатели без участия пациентов. Однако, некоторым пациентам вполне могли бы подойти более простые и дешевые устройства, которые пациенты могли бы использовать для записи необходимых показателей. Требуется разработать такое устройство.

Ваше устройство должно запрашивать измерение доступных ему показателей через заданное время (задается в минутах в прошивке устройства). Далее каждый интервал времени устройство должно запрашивать измерение одного из показателей – температуры или частоты пульса. О необходимости измерения пациента нужно оповещать звуком и частым морганием светодиода. Далее пациент должен использовать либо термодатчик, либо датчик пульса в зависимости от запроса устройства. Результат измерения устройство записывает в оперативную память. При измерении пульса, светодиод должен мерцать в такт ударам сердца. Приложение должно отображать результат последних измерений (температуры и пульса), сообщение о необходимости проведения замера (если пришло время провести замер) и иметь кнопку включения/отключения звуковых сигналов и запуска внепланового измерения всех показателей.

Датчики (сенсоры) для получения данных	**Устройства управления / отображения**
1) Датчик DS18B20	1) Светодиоды
2) Датчик пульса	2) Пьезоизлучатель

Вариант 8. Сортировка яблок на производстве

На современных производствах всё чаще и чаще используются системы технического зрения для различных целей. Они могут существенно помочь на разных этапах производства и сделать продукцию качественнее и дешевле. Часто это сложные и дорогие системы, но они не обязаны быть такими. И чтобы доказать это, вам нужно разработать устройство сортировки яблок на производстве. Устройство имеет датчик цвета, который позволяет определять цвет приближающихся к нему объектов. Датчику будут «показывать» яблоки, а он должен определять какого они цвета – зеленого или

красного. В зависимости от определенного цвета должен загораться зеленый или красный светодиод. Но если устройство видит, что цвет яблока больше коричневый, то, скорее всего, оно повреждено, и надо уведомить оператора линии коротким звуковым сигналом. Если поврежденных яблок становится больше 4-х подряд, то нужно остановить линию и включить сирену, чтобы обратить внимание обслуживающего персонала. Также на производстве должен жестко контролироваться уровень влажности. Если влажность превышает 70%, то должна включаться сирена. В приложении должны отображаться: количество яблок (общее и для каждого цвета отдельно) и уровень влажности на производстве. Также в нём должны присутствовать кнопки включения/отключения звуковых сигналов и сброса количества яблок.

Датчики (сенсоры) для получения данных	**Устройства управления / отображения**
1) Датчик цвета	1) Светодиоды
2) Датчик DHT11/22	2) Пьезоизлучатель

Вариант 9. Ультразвуковая рулетка

Современные технологии могут облегчить работу представителям совершенно разных профессий. Вам нужно разработать устройство, которое будет помогать бригадирам строительных бригад проверять труд своих подчиненных. Устройство должно иметь ультразвуковой датчик приближения и датчик цвета, дисплей и 2 кнопки. Одна кнопка должна включать измерение расстояния ультразвуковым датчиком. О завершении измерения устройство должно оповещать звуковым сигналом и морганием светодиода. Другая кнопка должна запускать измерение цвета (так бригадир будет контролировать правильный цвет покраски стен). Результаты измерений должны отображаться в приложении. Также в нём должны быть две кнопки: для выполнения измерения расстояния и цвета.

Датчики (сенсоры) для получения данных	**Устройства управления / отображения**
1) Ультразвуковой дальномер	1) Светодиоды
2) Тактовая кнопка (2 шт)	2) Пьезоизлучатель
3) Датчик цвета	

Вариант 10. Метеостанция

Всем нам так или иначе нравится/приходится следить за погодой. Но для того чтобы узнать погоду, не всегда хочется выходить на улицу. Поэтому нужно разработать устройство и приложение. Они предоставят нам нужную информацию и позволят не выходить на улицу. Если на улице идет дождь устройство должно включать светодиод, чтобы обратить внимание пользователя. Показатели с датчиков устройства должны выводиться в приложении на компьютере. Выводить нужно атмосферное давление, температуру и влажность. Также в приложении должны быть кнопки включения/отключения работы датчика атмосферного давления и световых сигналов.

Датчики (сенсоры) для получения данных	**Устройства управления / отображения**
1) Датчик BMP280	1) Светодиоды
2) Датчик температуры DHT22/11	
3) Датчик дождя	