



УНИВЕРСАЛЬНЫЙ ВЫЧИСЛИТЕЛЬНЫЙ КОНТРОЛЛЕР DXL-IOT



не для печати

Учебное пособие

ПРИКЛАДНАЯ
РОБОТОТЕХНИКА

УДК 004.896
ББК 32.816

ООО «Прикладная робототехника»

Универсальный вычислительный контроллер DXL-IoT / ООО «Прикладная робототехника» -
Электронная книга, 2021

Данное учебное пособие предназначено для применения совместно с универсальным вычислительным контроллером DXL-IoT, входящим в состав наборов производства ООО «Прикладная робототехника».

В данном учебном пособии разбирается процесс взаимодействия с контроллером, основы его программирования, и использования совместно с платами расширения его функционала.

Представленные в учебном пособии материалы будут полезны для изучения как педагогам, так и учащимся, желающим расширить базовые функциональные возможности популярных образовательных робототехнических наборов.

© ООО «Прикладная робототехника», 2021

ОГЛАВЛЕНИЕ

1. Обзор аппаратной составляющей.....	5
1.1. Вычислительный контроллер DXL-IoT.....	5
1.2. Плата расширения контроллера DXL-IoT с адаптером Ethernet ..	7
1.3. Силовая плата расширения контроллера DXL-IoT.....	9
2. Обзор программной составляющей.....	11
2.1. Подготовка среды разработки.....	11
2.2. Работа с Dynamixel - совместимыми устройствами ROBOTIS, библиотека DxlMaster.....	12
2.2.1. Инициализация библиотеки.....	13
2.2.2. Подключение произвольных устройств, класс DynamixelDevice	13
2.2.3. Подключение сервоприводов, класс DynamixelMotor	15
2.3. Работа модуля в качестве Dynamixel - совместимого устройства, библиотеки DxlSlave и DxlSlave2.....	17
2.3.1. Стандартная организация адресного пространства Dynamixel	17
2.3.2. Инициализация библиотеки.....	19
2.3.3. Работа с интерфейсом, класс DxlSlave	19
2.3.4. Примеры работы с библиотеками DxlSlave и DxlSlave2.....	20
3. Практическая часть.....	22
3.1. Управление встроенным светодиодом.....	22
3.2. Подключение УЗ-дальномера.....	23
3.3. Использование модуля беспроводной связи Bluetooth.....	25
3.4. Использование WiFi-адаптера.....	30
3.4.1. Работа в качестве WiFi клиента.....	30
3.4.2. Работа в качестве WiFi точки доступа.....	35
3.5. Использование платы расширения с адаптером Ethernet.....	37

3.6. Использование силовой платы расширения.....	39
3.7. Управление Dynamixel-совместимыми устройствами.....	42
3.7.1. Управление сервоприводами DYNAMIXEL.....	42
3.7.2. Управление Dynamixel-совместимыми периферийными модулями.....	44
3.7.3. Опрос Dynamixel-совместимого периферийного модуля.....	46
3.8. Конфигурирование контроллера, как Dynamixel-совместимое устройство.....	47
3.9. Управление мобильной платформой через Web-интерфейс.....	51

1. ОБЗОР АППАРАТНОЙ СОСТАВЛЯЮЩЕЙ

1.1. Вычислительный контроллер DXL-IoT

Одним из популярных вычислительных устройств, произведенных ООО «Прикладная робототехника» является программируемый контроллер DXL-IoT. Данное устройство представляет собой компактный вычислительный модуль, ориентированный на использование в образовательных проектах, где применяется технология «Интернета Вещей» (Internet of Things - IoT), благодаря тому, что он обладает компактными габаритами и необходимыми интерфейсами для подключения к сети. Помимо стандартного сетевого функционала, модуль обладает аппаратно реализованным Dynamixel - интерфейсом. Что позволяет контроллеру выполнять как роль мастера для взаимодействия с Dynamixel-совместимыми устройствами, так и, при необходимости, данный модуль может быть сконфигурирован таким образом, что сам будет являться Dynamixel-совместимым устройством. В качестве вычислительного микроконтроллера в универсальном вычислительном контроллере DXL-IoT используется микроконтроллер Atmega2560. Также на модуле размещен чип беспроводной связи по интерфейсам Wi-Fi и Bluetooth. (Рис. 1.1).

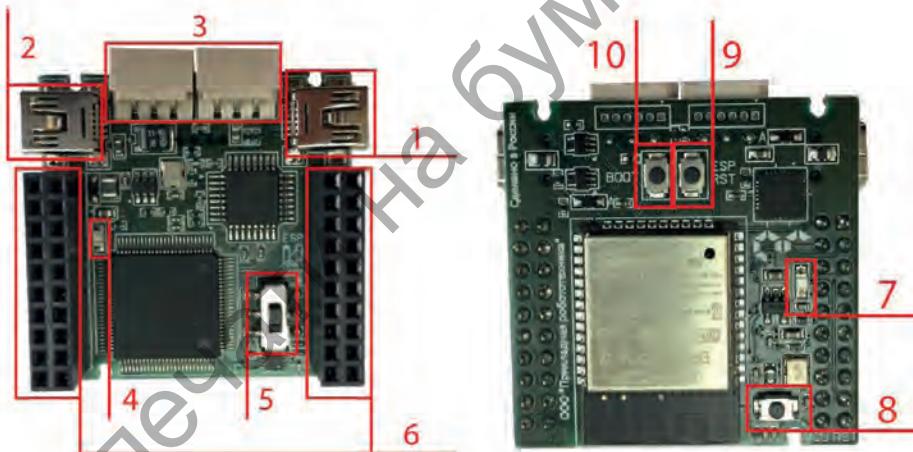


Рис. 1.1. Внешний вид вычислительного контроллера DXL-IoT спереди и сзади

Технические характеристики контроллера DXL-IoT выглядят следующим образом:

Габариты - 40x40 мм.

Напряжение питания, номинальное - 12В.

Flash-память - 256 KB (из которых 8 KB используются для загрузчика).
ОЗУ - 8 KB.

Энергонезависимая память - 4 KB.

Тактовая частота - 16 МГц.
Количество 3х пиновых портов Dynamixel - 2 шт.
Количество портов USB 2.0 - 2 шт.
Количество линий ввода-вывода - 40 шт. В том числе:
Линия «земля» - 1 шт.
Линия +3.3В - 1 шт.
Линия +5В - 1 шт.
Линия +12В - 1 шт.
Интерфейс SPI - 1 шт.
Интерфейс I2C - 1 шт.
Интерфейс UART - 1 шт.
Цифровые линии ввода-вывода - 12 шт.
Аналоговые линии ввода - 16 шт.
Индикаторы - 1 шт (5В).
Переключатель ESP-MCU - 1 шт.
Беспроводные интерфейсы:
Wi-Fi 802.11 b/g/n/d/e/i/k/r (802.11n до 150 Мбит/с),
Bluetooth V4.2 BR/EDR и BLE.
Органы управления (кнопки) - 3 шт.

Основные элементы контроллера следующие:

1. miniUSB порт для подключения к ПК и программирования основного контроллера - Atmega2560. Данный порт может быть использован для подачи на модуль питания 5В.
2. miniUSB порт для подключения к ПК и программирования модуля беспроводной связи Bluetooth / Wi-Fi. Данный порт может быть использован для подачи питания 5В на контроллер.
3. 3х пиновые DXL-порты. Используются для подключения контроллера в цепь в качестве Dynamixel-совместимого устройства. Контроллер DXL-IoT может быть использован и как ведущее Dynamixel-совместимое устройство (с его помощью можно управлять подключенными к нему сервоприводами DYNAMIXEL), так и как ведомое (в этом случае модуль выступает в качестве Dynamixel-совместимого устройства и обменивается данными с внешним контроллером через Dynamixel - интерфейс). 3х пиновые DXL-порты могут использоваться для подачи питания 12В на контроллер. При использовании контроллера в качестве Dynamixel-совместимого устройства, наличие внешнего источника питания 12В обязательно.
4. Индикационный светодиод линии 5В.
5. Переключатель интерфейса модуля беспроводного связи (Wi-Fi / Bluetooth модуля). В положении ESP (верхнее) интерфейс модуля беспроводной связи подключен к miniUSB порту для его программирования и настройки. В положении MCU (нижнее) интерфейс модуля беспроводной связи подключен к микроконтроллеру Atmega2560, что позволяет обмениваться с ним данными через Serial2 Atmega2560.

6. Гнезда с выводами для подключения внешних устройств и плат расширения.
7. Программируемый светодиод. Подключен к линии PB7 (D13) микроконтроллера Atmega2560.
8. Кнопка перезагрузки микроконтроллера Atmega2560.
9. Кнопка перезагрузки модуля беспроводной связи.
10. Кнопка ввода в режим загрузчика модуля беспроводной связи.

Распиновка боковых штыревых разъемов показана на рисунке 1.2:

12B	3.3V
PB0 (SS)	PB1 (SCK)
PB2 (MOSI)	PB3 (MISO)
PE4 (2 - PWM)	PE5 (3 - PWM)
PG5 (4 - PWM)	PE3 (5 - PWM)
PH3 (6 - PWM)	PH4 (7 - PWM)
PH5 (8 - PWM)	PH6 (9 - PWM)
PB4 (10 - PWM)	PB5 (11 - PWM)
PB6 (13 - PWM)	PB7 (13 - PWM)
PJ0 (RXD3)	PJ1 (TXD3)
GND	
5B	
PF0 (A0)	PF1 (A1)
PF2 (A2)	PF3 (A3)
PF4 (A4)	PF5 (A5)
PF6 (A6)	PF7 (A7)
PK0 (A8)	PK1 (A9)
PK2 (A10)	PK3 (A11)
PK4 (A12)	PK5 (A13)
PK6 (A14)	PK7 (A15)
PD0 (SCL)	PD1 (SDA)

Рис. 1.2. Распиновка боковых разъемов контроллера DXL-IoT

Для расширения функционала данного контроллера существует ряд модулей расширения.

1.2. Плата расширения контроллера DXL-IoT с адаптером Ethernet

Плата расширения контроллера DXL-IoT с адаптером Ethernet – вычислительный модуль, который, будучи установленным на контроллер DXL-IoT, позволяет выполнить подключение контроллера к сети с помощью интерфейса Ethernet, а также подключить и работать с картами памяти формата microSD (Рис. 1.3).

Основные элементы платы расширения DXL-IoT с адаптером Ethernet следующие:

1. Ethernet-разъем.
2. Индикационные светодиоды, сигнализирующие о работе сетевого соединения.
3. Разъем для установки microSD карты.
4. Кнопка для перезагрузки модуля.

Гнезда под штыревые разъемы, размещенные на данной плате расширения, являются сквозными для подключения к основному контроллеру DXL-IoT и его силовой плате расширения.

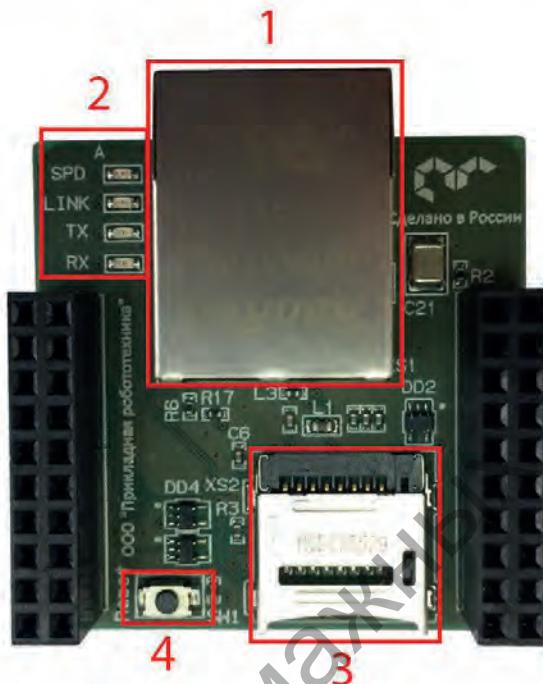


Рис. 1.3. Плата расширения контроллера DXL-IoT с адаптером Ethernet

Характеристики платы расширения DXL-IoT с адаптером Ethernet следующие:

Габариты - 40x40 мм.

Напряжение питания, номинальное - 5В.

Количество Ethernet разъемов - 1 шт.

Внутренний буфер - 16 КБ

Скорость подключения - 10/100 Мбит/с

Количество одновременных подключений - 4 шт.

Поддержка протоколов - TCP, UDP, ICMP, IPv4 AR

Режимы – полудуплекс и полный дуплекс.

Интерфейс SPI - 1 шт.

Количество microSD разъемов - 1 шт.

Поддержка карт памяти microSD, объемом до 16GB.

Индикаторы - 4 шт. (SPD, LINK, RX, TX)

Органы управления (кнопки) - 1 шт.

Данный вычислительный модуль полностью совместим со стандартными библиотеками и примерами в Arduino IDE, применяемыми для работы с Ethernet-адаптерами, разработанными на чипах W5100 или W5500.

1.3. Силовая плата расширения контроллера DXL-IoT

Для управления силовой нагрузкой, такой как двигатели постоянного тока или источники освещения, существует специальная плата расширения к контроллеру DXL-IoT с возможностью коммутации больших токов и напряжений. Внешний вид силовой платы расширения контроллера DXL-IoT представлен на Рис. 1.4.

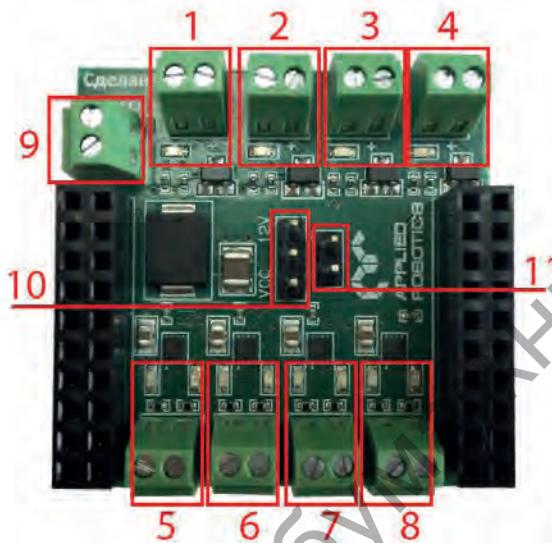


Рис. 1.4. Силовая плата расширения контроллера DXL-IoT

Основные элементы силовой платы расширения контроллера DXL-IoT следующие:

1. Клемник для управления силовой нагрузкой и соответствующий ему индикационный светодиод. Управляется линией PB4 (D10).
2. Клемник для управления силовой нагрузкой и соответствующий ему индикационный светодиод. Управляется линией PB5 (D11).
3. Клемник для управления силовой нагрузкой и соответствующий ему индикационный светодиод. Управляется линией PB6 (D12).
4. Клемник для управления силовой нагрузкой и соответствующий ему индикационный светодиод. Управляется линией PB7 (D13).
5. Клемник для подключения нагрузки в виде двигателя постоянного тока и управления им, путем задания направления вращения и скорости. Управляется линиями PE4 (D2) - направление, и PE5 (D3) - скорость. Индикационные светодиоды показывают направление вращения.
6. Клемник для подключения нагрузки в виде двигателя постоянного тока и управления им, путем задания направления вращения и скорости. Управляется линиями PG5 (D4) - направление, и PE3 (D5) - скорость. Индикационные светодиоды показывают направление вращения.
7. Клемник для подключения нагрузки в виде двигателя постоянного тока и управления им, путем задания направления вращения и скоро-

сти. Управляется линиями PH3 (D6) - направление, и PH4 (D7) - скорость. Индикационные светодиоды показывают направление вращения.

8. Клемник для подключения нагрузки в виде двигателя постоянного тока и управления им, путем задания направления вращения и скорости. Управляется линиями PH5 (D8) - направление, и PH6 (D9) - скорость. Индикационные светодиоды показывают направление вращения.

9. Клемник для подключения внешнего питания. В зависимости от типа двигателя постоянного тока, может разниться напряжение питания, необходимое для его корректной работы.

10. Перемыкатель, с помощью которого можно выбрать источник питания для коммутации напряжения и управления двигателями. В положении 12V, источником питания является 12В с контроллера DXL-IoT. Которые тот, в свою очередь, получает через 3х пиновые DXL разъемы. В положении VCC источником питания является источник, подключенный через клемник 9. Таким образом, в положении VCC контроллер может питаться от 12В или 5В, а силовая плата расширения работать с другими напряжениями, например, 9В.

11. Перемыкатель, объединяющий линию 12В и VCC от клемника 9. Позволяет использовать клемник 9, как источник питания 12В. Также, в свою очередь, такая схема подключения может быть использована для питания контроллера напряжением от клемника 9, например, 9В. Но при этом к 3х пиновым DXL выводам контроллера не должно быть подведено 12В.

Характеристики силовой платы расширения контроллера DXL-IoT следующие:

Габариты - 40x40мм.

Напряжение питания - 5 - 12В.

Количество линий для коммутации напряжения - 4 шт.

Максимальное напряжение коммутации - 12В.

Допустимый ток коммутации на 1 линию - 2А.

Количество линий управления двигателями постоянного тока - 4 шт.

Количество индикационных светодиодов - 12 шт.

Все рассмотренные платы расширения к контроллеру DXL-IoT могут быть собраны в одно устройство, способное одновременно управлять сервоприводами DYNAMIXEL, быть подключенным к сети и выполнять какую-либо коммутацию. Единственное, о чем необходимо помнить и что нужно контролировать - каждая плата расширения использует в своих целях некоторые из доступных линий, идущих от основного контроллера. Тем самым, функционал платы расширения может быть уменьшен. Например, плата расширения с адаптером Ethernet использует линию PB4 (D10), как «chip select» для микросхемы Ethernet. В результате чего, при одновременной работе платы расширения с адаптером Ethernet и силовой платы расширения, на последней не будет управляться клемник 1, который управляется этой же линией.

2. ОБЗОР ПРОГРАММНОЙ СОСТАВЛЯЮЩЕЙ

2.1. Подготовка среды разработки

Универсальный вычислительный контроллер DXL-IoT является подобной Arduino Mega 2560 платой. С помощью среды Arduino IDE для него можно разрабатывать программный код, применяя те же подходы и библиотеки, что и для Arduino Mega 2560. Для работы с контроллером рекомендуется использовать среду разработки Arduino IDE версии не ниже 1.8, которую необходимо скачать с сайта Arduino.cc и инсталлировать в операционную систему штатными средствами. Поскольку на контроллере имеется модуль беспроводной связи Wi-Fi / Bluetooth, необходимо добавить в среду разработки поддержку данного модуля. Для этого нужно перейти в меню «Файл» - «Настройки» - «Дополнительные ссылки для Менеджера плат» и в открывшемся окне (Рис. 2.1) вставить следующую ссылку:

https://dl.espressif.com/dl/package_esp32_index.json

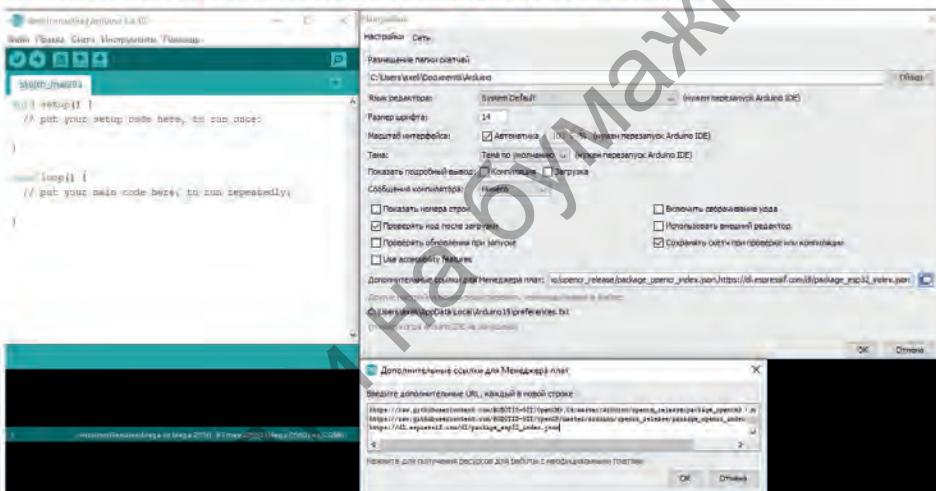


Рис. 2.1. Добавление ссылки для менеджера плат

В обоих окнах нажать кнопку «OK».

После указания дополнительной ссылки для менеджера плат требуется перейти в раздел «Инструменты» - «Платы» - «Менеджер плат», в строке поиска ввести esp32 и установить найденный пакет «esp32 by Espressif Systems» (Рис. 2.2).

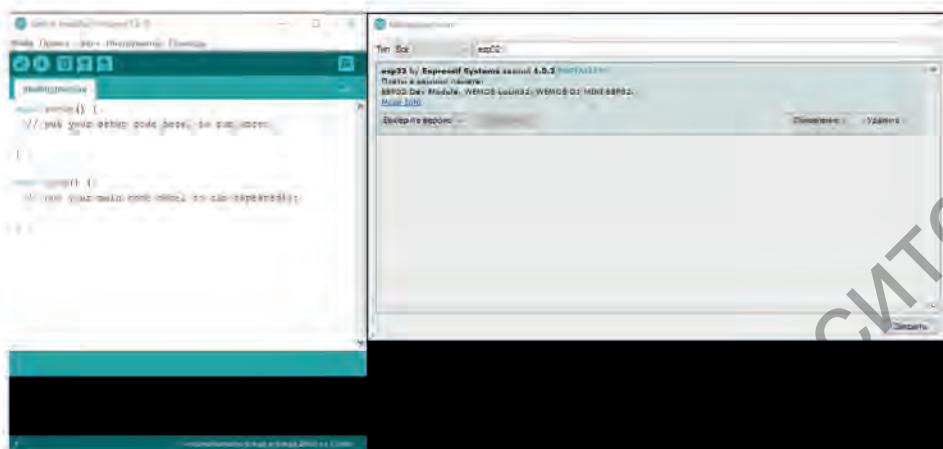


Рис. 2.2. Установка пакета поддержки плат семейства ESP32

В результате этих действий, в меню «Инструменты» - «Плата» появится раздел «ESP32 Arduino», в котором для работы со встроенным модулем беспроводной связи WiFi / Bluetooth необходимо выбрать «ESP32 Dev Module».

Отличительной особенностью контроллера DXL-IoT является поддержка интерфейса Dynamixel, для работы с которым разработаны специальные библиотеки. Библиотеки для работы с Dynamixel-протоколом можно скачать с сайта appliedrobotics.ru из раздела «Учебные материалы» - «Программное обеспечение». Существует 2 способа работы контроллера в цепи Dynamixel – совместимых устройств: в качестве «Master-устройства» (ведущий контроллер) и «Slave-устройства» (ведомое Dynamixel-совместимое устройство).

2.2. Работа с Dynamixel – совместимыми устройствами, библиотека DxIMaster

Dynamixel – совместимые устройства – сервоприводы и датчики, поддерживающие протокол Dynamixel 1.0 или 2.0, подключаются к интерфейсу на основе последовательного порта UART в режиме полудуплекса. Для управления устройствами Dynamixel с универсального вычислительного контроллера DXL-IoT необходимо использовать библиотеку «DxIMaster», загружаемую с сайта компании ООО «Прикладная робототехника» (appliedrobotics.ru). Данная библиотека поставляется в виде zip-библиотеки Arduino и устанавливается штатными средствами Arduino IDE (Меню «Скетч» -> «Подключить библиотеку» -> «Добавить .ZIP библиотеку»).

Библиотека «DxIMaster» содержит следующие примеры (меню «Файл» -> «Примеры»):

- `blink_led` – включение и выключение светодиодов сервоприводов

DYNAMIXEL.

- **change_baudrate** - изменение скорости передачи данных на шине Dynamixel.
- **console** - терминал, позволяющий из командной строки найти и настроить Dynamixel - совместимые устройства в сети.
- **joint_mode** - управление сервоприводами по положению (режим шарнирного сочленения).
- **wheel_mode** - управление сервоприводами по скорости (режим колеса).
- **wheel_mode_low_level** - управление сервоприводами на низком уровне (изменением значений регистров).

2.2.1. Инициализация библиотеки

Во всех случаях использования библиотеку «DxlMaster» необходимо добавить в программу и инициализировать. Для добавления библиотеки необходимо добавить следующий заголовочный файл в начале программы:

```
#include <DxlMaster.h>
```

При добавлении заголовочного файла библиотеки становится доступен глобальный объект DxlMaster, который необходимо инициализировать следующим образом:

```
DxlMaster.begin(int baudrate);
```

где **baudrate** – скорость обмена данными с устройствами на шине Dynamixel (см. документацию к конкретной модели привода). Чаще всего используются скорости 57 600 бод и 1 000 000 бод. Скорость обмена данными с Dynamixel – совместимым устройством может быть изменена через программу ROBOTIS Dynamixel Wizard, либо используя пример `change_baudrate`.

2.2.2. Подключение произвольных устройств, класс *DynamixelDevice*

Обмен данными с Dynamixel – совместимыми устройствами происходит путем записи и чтения регистров в памяти такого устройства. Для этого предназначен класс *DynamixelDevice* библиотеки «DxlMaster».

Общие методы класса DxlMaster:

- **DxlMaster(uint8_t id)** – конструктор, **id** – уникальный номер устройства Dynamixel в сети (см. документацию на устройство). Может быть изменен в программе ROBOTIS Dynamixel Wizard.

- **DynamixelStatus init()** – поиск и инициализация устройства. Если найдено устройство с адресом **id** и скоростью обмена, соответствующей теч-

кущим настройкам шины, возвращает 0, иначе - код ошибки.

- **DynamixelStatus changeId(uint8_t id)** - изменение id устройства (для этого устройство все равно должно быть инициализировано с исходным id).

- **uint16_t model()** - возвращает номер модели устройства.

- **uint8_t firmware()** - возвращает версию прошивки устройства.

- **void communicationSpeed(uint32_t aSpeed)** - изменение скорости обмена данными с устройством.

Обмен данными:

Все методы для обмена данными возвращают статус выполнения команды DynamixelStatus, принимающий значение «0» в случае успеха или выдающий номер стандартной ошибки шины Dynamixel, в случае сбоя обмена данными.

Методы чтения данных выполнены в виде шаблонов для чтения регистров разной длины (чаще всего в Dynamixel - совместимых устройствах применяются регистры типов uint8_t, uint16_t), например:

`template<class T> DynamixelStatus read(uint8_t aAddress, T& aData)`

- шаблон метода для чтения регистра произвольного типа **T** по адресу **aAddress**.

Пример использования:

```
uint16_t pos;
device.read(DYN_ADDRESS_CURRENT_POSITION, pos);
```

После успешного выполнения команды в переменной **pos** окажется значение 16-ти разрядного регистра **DYN_ADDRESS_CURRENT_POSITION**, который занимает в памяти сервопривода 2 байта.

Методы для обмена данными:

- **DynamixelStatus read(uint8_t aAddress, T& aData)** - чтение регистра типа **T** по адресу **aAddress** с записью результата в переменную **aData**.

- **DynamixelStatus read(uint8_t aAddress, uint8_t size, uint8_t *ptr)** - чтение массива из регистров, в количестве **size** по адресу **aAddress** с записью результата в массив по адресу **ptr**.

- **DynamixelStatus write(uint8_t aAddress, const T& aData)** - запись в регистр типа **T** по адресу **aAddress** значения **aData**.

- **DynamixelStatus write(uint8_t aAddress, uint8_t size, const uint8_t *ptr)** - запись в массив регистров, длиной **size**, по адресу **aAddress** значений из локального массива с указателем **ptr**.

- **DynamixelStatus regWrite(uint8_t aAddress, const T& aData)** - команда отложенной записи в регистр типа **T** по адресу **aAddress** значения **aData**. Значение передается на устройство, но применяется к регистру только после команды **action()**.

- **DynamixelStatus regWrite(uint8_t aAddress, uint8_t size, const uint8_t**

***ptr**) - команда отложенной записи в массив регистров, длиной **size**, по адресу **aAddress** значений из локального массива по указателю **ptr**.

- **DynamixelStatus action()** - команда, получив которую все устройства в сети применяют значения регистров, полученные с командой отложенной записи. Функция применяется для организации синхронной работы сервоприводов: вначале всем сервоприводам рассылаются целевые значения (положение, скорость), а затем широковещательной командой **«action»** приводы запускаются.

- **DynamixelStatus ping()** - команда для проверки наличия устройства в сети. В случае, если устройство найдено, возвращает значение «0», иначе - код ошибки.

Примечание. Для отправки широковещательных команд необходимо создать абстрактное устройство **DynamixelDevice** с широковещательным адресом.

Пример низкоуровневого управления сервоприводом DYNAMIXEL MX-28:

```
#include «DxlMaster.h» // Добавление библиотеки
DynamixelDevice device(1); // Создание объекта управления с адресом
id = 1
void setup() {
    DxlMaster.begin(1000000); // Инициализация шины со скоростью
1000000 бод
    device.init(); // Поиск в сети и инициализация устройства
    device.write(DYN_ADDRESS_ENABLE_TORQUE, 1); // Включение момента
    uint16_t zero_limit = 0;
    device.write(DYN_ADDRESS_CW_LIMIT, zero_limit); // Очистка ограничения вращения по часовой стрелке
    device.write(DYN_ADDRESS_CCW_LIMIT, zero_limit); // Очистка ограничения вращения против часовой стрелки
}
void loop() {
    int16_t speed = 1000;
    device.write(DYN_ADDRESS_GOAL_SPEED, speed); // Установка скоро-
сти
}
```

2.2.3. Подключение сервоприводов, класс **DynamixelMotor**

Для более удобного управления сервоприводами DYNAMIXEL, необходимо применять ориентированный на такое использование класс **DynamixelMotor**, наследующий класс **DynamixelDevice** вместе со всеми представленными в предыдущем разделе методами. Дополнительно к

методам класса **DynamixelDevice** в классе **DynamixelMotor** реализованы следующие специальные методы.

- **void wheelMode()** - настраивает регистры привода для работы в режиме колеса.

- **void jointMode(uint16_t aCWLlimit=0, uint16_t aCCWLlimit=0x3FF)** - настраивает привод для управления по положению (режим шарнирного сочленения) с ограничениями углов поворота по и против часовой стрелки (**aCWLlimit** и **aCCWLlimit**, соответственно).

- **void enableTorque(bool aTorque=true)** - включает момент, регистр целевого положения принимает значение текущего положения, и сервопривод удерживает текущее положение.

- **void speed(int16_t aSpeed)** - задает скорость вращения привода в режиме колеса, либо максимальную скорость вращения в режиме сочленения.

- **void goalPosition(uint16_t aPosition)** - задает целевое положение при управлении по положению.

- **void led(uint8_t aState)** - управляет светодиодом сервопривода: **aState = 0** - выключение, **aState = 1** - включение.

- **uint16_t currentPosition()** - опрашивает и возвращает показания о текущем угловом положении сервопривода.

- **DynamixelStatus getCurrentPosition(uint16_t &pos)** - метод аналогичен предыдущему, но возвращает статус выполнения команды для построения более надежных систем управления.

Пример управления мотором:

```
#include «DxlMaster.h» // Добавление библиотеки
DynamixelMotor motor(1); // Создание объекта класса сервопривода с
адресом 1
void setup(){
    DxlMaster.begin(1000000); // Инициализация шины Dynamixel со скоро-
стю 1000000 бод
    motor.enableTorque(); // Включение момента привода
    motor.jointMode(204, 820); // Включение режима сочленения с угловы-
ми ограничениями
    motor.speed(100); // Задание максимальной скорости вращения при-
вода
}
void loop(){
    motor.goalPosition(123); // Задание положения привода
}
```

Более подробно возможности библиотеки представлены в примерах среды Arduino IDE (Меню «Файл» -> «Примеры» -> «DxlMaster»).

2.3. Работа модуля в качестве Dynamixel - совместимого устройства, библиотеки DxISlave и DxISlave2

Модуль подключается в сеть устройств Dynamixel в качестве ведомого, отвечает на запросы по протоколам Dynamixel 1.0 и Dynamixel 2.0 и передает данные на управляющий контроллер более высокого уровня через интерфейс на основе последовательного порта UART в режиме полудуплекса (Рис.1, поз. 16).

Для работы интерфейса и предоставления доступа к данным для опроса по протоколам Dynamixel 1.0 и 2.0 необходимо использовать библиотеки «DxISlave» и «DxISlave2.0», которые можно загрузить с сайта компании ООО «Прикладная робототехника» (из раздела «Учебные материалы»). Библиотеки поставляются в виде zip-библиотек Arduino и устанавливаются штатными средствами Arduino IDE (Меню «Скетч» -> Подключить библиотеку -> Добавить ZIP библиотеку).

Библиотека содержит следующие примеры (меню «Файл» -> Примеры):

- **allRegs** – организация Dynamixel-совместимого устройства с произвольным набором данных.
- **setEepromDefault** – очистка энергонезависимой памяти, используемой для хранения значений регистров Dynamixel.
- **ax12a** (только в DxISlave) – эмуляция сервопривода ROBOTIS DYNAMIXEL AX-12A.
- **xl320** (только в DxISlave2) – эмуляция сервопривода ROBOTIS DYNAMIXEL XL-320.

2.3.1. Стандартная организация адресного пространства Dynamixel

В библиотеках организовано регистровое управление, аналогичное устройствам Dynamixel – данные, доступные внешним устройствам, необходимо хранить в выделенной области памяти библиотеки в 255 байт (регистров), каждый из которых имеет адрес от 0 до 255.

Некоторые регистры по стандартным адресам должны содержать типовые для Dynamixel-совместимых устройств значения и заполняются библиотекой автоматически, другие отведены для пользовательских задач. Адресное пространство и назначение регистров приведено в таблице 1.

Область	Адрес (HEX)	Имя	Описание	Доступ	Начальное значение
EEPROM	0 (0x00)	Model Number(L)	Младший байт номера модели устройства	R*	12 (0x0C)
	1 (0x01)	Model Number(H)	Старший байт номера модели устройства	R	0 (0x00)
	2 (0x02)	Version of Firmware	Версия программного обеспечения устройства	R	-
	3 (0x03)	ID	Адрес устройства в сети Dynamixel	RW	1 (0x01)
	4 (0x04)	Baud Rate	Скорость обмена данными (делитель). 1 - 1000000 бод	RW	1 (0x01)
	5 (0x05)	Return Delay Time	Задержка ответа на запросы для повышения надежности передачи данных	RW	250 (0xFA)
	6 (0x06)		Произвольные данные	RW	0(0x00)

	15 (0x0F)		Произвольные данные	RW	0(0x00)
	16 (0x10)	Status Return Level	Настройка протокола: будет ли устройство отвечать на запросы	RW	2 (0x02)
	17 (0x11)		Произвольные данные	RW	0(0x00)

	23 (0x17)		Произвольные данные	RW	0(0x00)
	24 (0x18)		Произвольные данные	RW	0(0x00)
RAM
	255 (0xFF)		Произвольные данные	RW	0(0x00)

Таблица 1. Назначение регистров Dynamixel

* R - внешним устройствам доступно только чтение регистра, RW - чтение и запись.

2.3.2. Инициализация библиотеки

Во всех случаях использования библиотеку необходимо добавить в программу и инициализировать. Для добавления библиотеки необходимо подключить заголовочный файл в начале программы.

Для работы с протоколом Dynamixel 1.0:

```
#include «DxISlave.h»
```

Для работы с протоколом Dynamixel 2.0:

```
#include «DxISlave2.h»
```

При добавлении заголовочного файла библиотеки становится доступен глобальный объект DxISlave (одинаковый для любой версии протокола), который необходимо инициализировать следующим образом:

```
DxISlave.begin(uint16_t model_number, uint8_t fw_version);
```

где `model_number` - номер модели устройства и `fw_version` - версия программного обеспечения, которые будут показаны внешним устройствам.

2.3.3. Работа с интерфейсом, класс DxISlave

После инициализации библиотека работает в фоновом режиме и автоматически отвечает на запросы внешних устройств, отправляя данные из регистров или устанавливая их новые значения. Изменение значений системных (стандартных) регистров с настройками параметров передачи данных (ID устройства, baud rate, return delay time, status return level) автоматически приводит к изменению этих параметров. Изменение значений пользовательских регистров фиксируется в виде их адресов в кольцевом буфере (очереди). Таким образом, обмен данными необходимо выполнять в следующей последовательности:

- чтобы предоставить данные управляющему устройству необходимо записать их в регистр, и они автоматически будут отправлены управляющему устройству по запросу;

- чтобы узнать об изменениях регистров, выполненных управляющим устройством, и произвести соответствующие действия, необходимо проверить наличие необработанных изменений в очереди (polling).

Для работы с интерфейсом необходимо использовать класс DxISlave, предоставляющий следующие методы.

- `void begin(uint16_t model_number, uint8_t fw_version)` - инициализация библиотеки. Значения `model_number` и `fw_version` записываются в регистры **Model Number** и **Version of Firmware** в соответствии с табли-

цей стандартных регистров. После вызова метода, устройство начинает отвечать и автоматически (в фоновом режиме) реагировать на запросы Dynamixel.

- `void set_id(uint8_t id)` - записывает **ID** в соответствии с таблицей регистров. После этого устройство автоматически меняет адрес доступа.

- `void set_baud(uint8_t baud)` - записывает **Baud Rate** в соответствии с таблицей регистров. После этого устройство меняет скорость передачи данных (**baud rate**).

- `void set(uint8_t addr, uint8_t val)` - записывает значение **val** в регистр по адресу **addr**. Значение не перезаписывается в EEPROM, если равно уже записанному. При изменении значений регистров, отвечающих за настройки протокола, автоматически меняются соответствующие параметры передачи данных.

- `uint8_t get(uint8_t addr)` - возвращает значение, записанное в регистре по адресу **addr**.

- `void set_mode(uint8_t addr, uint8_t mode)` - настраивает режим доступа **mode** (0 - R (чтение), 1 - RW (чтение и запись)) к регистру по адресу **addr**.

- `void set_mode16(uint8_t addr, uint8_t mode)` - настраивает режим доступа к 2-м регистрам подряд начиная с адреса **addr**.

- `void set16(uint8_t addr, uint16_t val)` - записывает 16-ти разрядное значение в 2 регистра, расположенных по адресу **addr**. Последовательность байт: младший байт первый. Значение не должно перезаписываться в EEPROM, если равно уже записанному.

- `uint16_t get16(uint8_t addr)` - возвращает 16-ти разрядное значение, записанное в 2-х регистрах (первый байт младший).

- `uint8_t poll()` - возвращает 1, если со стороны ведущего устройства через протокол Dynamixel было изменено значение какого-либо регистра (очередь измененных регистров не пуста); 0, если изменений нет (очередь пуста).

- `uint8_t scan()` - возвращает адрес первого в очереди (кольцевом буфере) измененного через протокол Dynamixel регистра и удаляет его из очереди (смещает указатель кольцевого буфера на следующий измененный регистр).

2.3.4. Примеры работы с библиотеками *DxlSlave* и *DxlSlave2*

В следующем примере показана организация Dynamixel-совместимого устройства с произвольным набором данных. В этом примере при изменении ведущим устройством регистра разрабатываемого ведомого устройства, ведомое устройство выводит в терминал адрес и значение измененного регистра.

```

#include <DxlSlave.h> // Подключение библиотеки
#define IS_INITED_REG 23 // Адрес регистра для проверки 1-го запуска
void setup(){
    DxlSlave.begin(1234, 12); // Инициализация интерфейса (номер модели
    1234, версия 12)
    Serial.begin(9600); // Инициализация отладочного последовательного
    порта для вывода
    if(!DxlSlave.get(IS_INITED_REG)){ // Инициализация EEPROM при 1-м за-
    пуске
        for(int i = 6; i < 23; i++){
            DxlSlave.set(i, i); // Заполнение регистров EEPROM произвольными
            значениями
            DxlSlave.set(IS_INITED_REG, 1); // Установка флага «1-я инициализация
            выполнена»
        }
    }
    for(int i = 24; i < 255; i++){
        DxlSlave.set(i, i); // Заполнение регистров RAM произвольными значе-
       ниями
    }
}
void loop(){
    while(DxlSlave.poll()){ // Проверка наличия изменений регистров веду-
    щим устройством
        uint8_t addr = DxlSlave.scan(); // Получение адреса измененного реги-
       стра из очереди
        Serial.print(" The value at address "); // Вывод информации
        Serial.print(addr);
        Serial.print(" was changed by master. New value is ");
        Serial.println(DxlSlave.get(addr));
    }
}

```

Подробнее возможности библиотеки представлены в примерах
(Меню «Файл»-> Примеры -> DxlSlave).

Для очистки EEPROM устройства, в том числе, для очистки флага и
вызыва 1-й инициализации EEPROM при изменении программы, необхо-
димо использовать пример setEepromDefault.

Организация произвольного Dynamixel-совместимого устройства с
использованием более разнообразных команд приведена в стандартном
примере allRegs библиотеки.

Возможности библиотеки DxlSlave продемонстрированы также в
примере ax12a, где эмулируется сервопривод ROBOTIS DYNAMIXEL AX-
12, распознаваемый стандартными утилитами ROBOTIS.

Аналогичный пример эмуляции сервопривода XL-320 с протоколом
Dynamixel 2.0 приведен в библиотеке DxlSlave2.

3. ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1. Управление встроенным светодиодом

На универсальном вычислительном контроллере DXL-IoT имеется светодиод (позиция №7, линия управления которого заведена на линию PB7 (D13) управляющего микроконтроллера Atmega2560). Такой светодиод имеется у большинства Arduino-совместимых плат и может быть использован как для тестирования работоспособности микроконтроллера, так и для индикации состояния выполнения программы. Для управления им можно воспользоваться простейшим скетчем, приведенном в примере: «Файлы» -> «Примеры» -> «01.Basic» -> «Blink»:

```
void setup(){
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop(){
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

В результате загрузки кода данного примера в контроллер, светодиод начнет мигать с периодичностью в 1 секунду (Рис. 3.1).

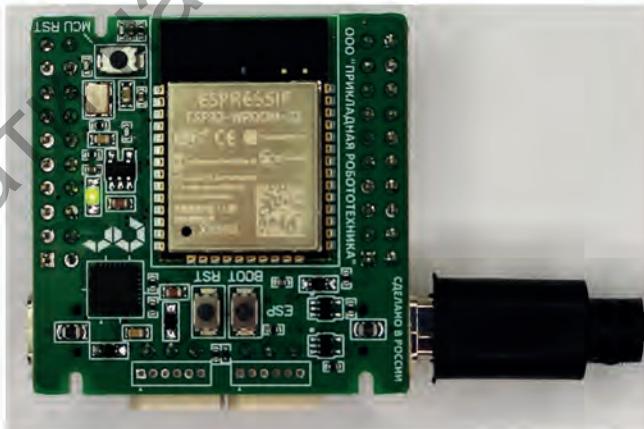


Рис. 3.1. Управление встроенным светодиодом

3.2. Подключение УЗ-дальномера

Поскольку программируемый контроллер DXL-IoT является Arduino-подобной платформой, то к нему можно подключать популярные Arduino-совместимые датчики таким же образом, как они подключаются к платам типа Arduino Mega2560 или Arduino Uno. В качестве примера рассмотрим взаимодействие с типовым датчиком расстояния. Проводное подключение УЗ дальномера к контроллеру (Рис. 3.2) выполняется в соответствии со схемой распиновки контроллера, показанной на рисунке 1.2:

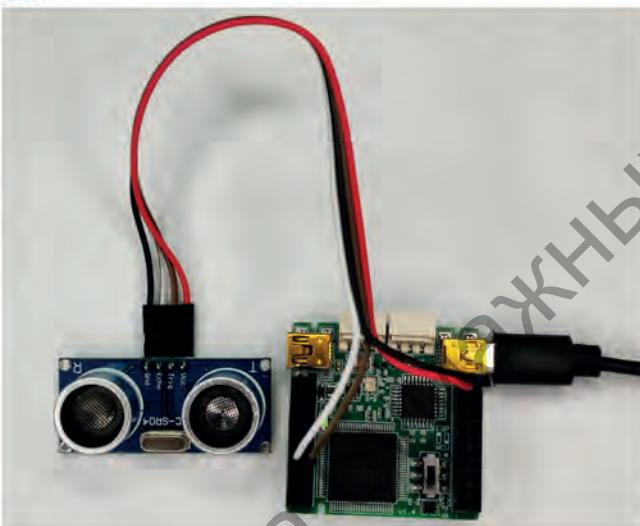


Рис. 3.2. Подключение УЗ-дальномера к контроллеру DXL-IoT

Код для опроса данных ультразвукового датчика расстояния выглядит стандартным образом:

```
#define PIN_TRIGGER 9
#define PIN_ECHO 8

long duration, cm;

void setup(){
    // Инициализируем взаимодействие по последовательному порту
    Serial.begin(115200);
    //Определяем вводы и выводы
    pinMode(PIN_TRIGGER, OUTPUT);
    pinMode(PIN_ECHO, INPUT);
}
```

```
void loop() {  
    // Сначала генерируем короткий импульс длительностью 2-5 микросекунд.  
  
    digitalWrite(PIN_TRIGGER, LOW);  
    delayMicroseconds(5);  
    digitalWrite(PIN_TRIGGER, HIGH);  
  
    // Выставив высокий уровень сигнала, ждем около 10 микросекунд. В этот момент датчик будет посылать сигналы с частотой 40 КГц.  
    delayMicroseconds(10);  
    digitalWrite(PIN_TRIGGER, LOW);  
  
    // Время задержки акустического сигнала на эхолокаторе:  
    duration = pulseIn(PIN_ECHO, HIGH);  
  
    // Теперь осталось преобразовать время в расстояние  
    cm = (duration / 2) / 29.1;  
  
    Serial.print("Расстояние до объекта:");  
    Serial.print(cm);  
    Serial.println(" см.");  
  
    // Задержка между измерениями для корректной работы скетча  
    delay(250);  
}
```

В результате, в монитор порта будет выведена следующая информация (Рис. 3.3):

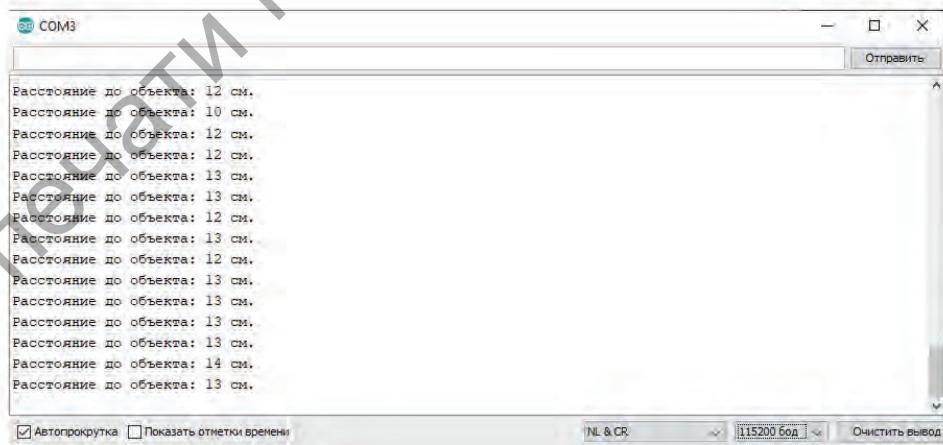


Рис. 3.3. Вывод данных в монитор морта

3.3. Использование модуля беспроводной связи Bluetooth

Модуль беспроводной связи, установленный на контроллере DXL-IoT может работать как в режиме Wi-Fi, так и в режиме Bluetooth 4 (он же Bluetooth LE или BLE). Для активации требуемого режима работы необходимо загрузить определенный программный код в модуль с помощью среды разработки Arduino IDE.

В качестве основного примера для настройки работы модуля в качестве Bluetooth устройства будет использоваться стандартный пример BLE_uart. Данный пример находится, по умолчанию, после установки программной поддержки модуля в среде Arduino IDE в меню «Файл» -> «Примеры» -> ESP32 BLE Arduino -> BLE_uart. Для отображения данного примера в среде Arduino IDE должна быть выбрана плата ESP32 Dev Module. Данный пример необходимо модернизировать таким образом, чтобы задать необходимое наименование нашего BLE-устройства и добавить код, передающий получаемые по Bluetooth каналу данные в вычислительный микроконтроллер atmega2560.

Для изменения наименования устройства необходимо внести изменения в строку

```
BLEDevice::init("UART Service");
```

где вместо «UART Service» необходимо указать требуемое наименование, например, «My Robot»:

```
BLEDevice::init("My Robot");
```

В результате, код будет выглядеть следующим образом:

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

BLEServer *pServer = NULL;
BLECharacteristic * pTxCharacteristic;
bool deviceConnected = false;
bool oldDeviceConnected = false;
uint8_t txValue = 0;

// See the following for generating UUIDs:
// https://www.uuidgenerator.net/

#define SERVICE_UUID      «6E400001-B5A3-F393-E0A9-E50E24DCCA9E»
// UART service UUID
#define CHARACTERISTIC_UUID_RX    «6E400002-B5A3-F393-E0A9-
E50E24DCCA9E»
#define CHARACTERISTIC_UUID_TX    «6E400003-B5A3-F393-E0A9-
E50E24DCCA9E»
```

```
class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
        deviceConnected = true;
    }

    void onDisconnect(BLEServer* pServer) {
        deviceConnected = false;
    }
};

class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        std::string rxValue = pCharacteristic->getValue();

        if (rxValue.length() > 0) {
            Serial.println("*****");
            Serial.print("Received Value: ");
            for (int i = 0; i < rxValue.length(); i++)
                Serial.print(rxValue[i]);

            Serial.println();
            Serial.println("*****");
        }
    }
}

void setup() {
    Serial.begin(115200);

    // Create the BLE Device
    BLEDevice::init("My Robot");

    // Create the BLE Server
    pServer = BLEDevice::createServer();
    pServer->setCallbacks(new MyServerCallbacks());

    // Create the BLE Service
    BLEService *pService = pServer->createService(SERVICE_UUID);

    // Create a BLE Characteristic
    pTxCharacteristic = pService->createCharacteristic(
        CHARACTERISTIC_UUID_TX,
        BLECharacteristic::PROPERTY_NOTIFY
    );

    pTxCharacteristic->addDescriptor(new BLE2902());
}
```

```

BLECharacteristic * pRxCharacteristic = pService->createCharacteristic(
    CHARACTERISTIC_UUID_RX,
    BLECharacteristic::PROPERTY_WRITE
);

pRxCharacteristic->setCallbacks(new MyCallbacks());

// Start the service
pService->start();

// Start advertising
pServer->getAdvertising()->start();
Serial.println(«Waiting a client connection to notify...»);
}

void loop(){

if (deviceConnected) {
    pTxCharacteristic->setValue(&txValue, 1);
    pTxCharacteristic->notify();
    txValue++;
    delay(10); // bluetooth stack will go into congestion, if too
many packets are sent
}

// disconnecting
if (!deviceConnected && oldDeviceConnected) {
    delay(500); // give the bluetooth stack the chance to get things ready
    pServer->startAdvertising(); // restart advertising
    Serial.println(«start advertising»);
    oldDeviceConnected = deviceConnected;
}
// connecting
if (deviceConnected && !oldDeviceConnected){
    // do stuff here on connecting
    oldDeviceConnected = deviceConnected;
}
}

```

Для загрузки данного кода в модуль беспроводной связи необходимо подключить его к компьютеру через соответствующий miniUSB порт, переместить переключатель в положение «ESP», нажать в среде Arduino IDE кнопку загрузки и, по завершении процесса компиляции, когда в терминал будет выведена строка «Connecting....» (Рис. 3.4), зажать на 2-3 секунды обе кнопки, затем отпустить RST и потом отпустить Boot на модуле.

The screenshot shows the Arduino IDE interface. The top menu bar includes 'Файл', 'Правка', 'Скетч', 'Инструменты', and 'Помощь'. The title bar says 'BLE_uart | Arduino 1.8.10'. The code editor contains the following C++ code:

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

BLEServer *pServer = NULL;
BLECharacteristic * pTxCharacteristic;
bool deviceConnected = false;
bool oldDeviceConnected = false;
uint8_t txValue = 0;

// See the following for generating UUIDs:
// https://www.uuidgenerator.net/

#define SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCAE" // UART service UUID
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCAE"
#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCAE"

class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
        deviceConnected = true;
    }
};

Загрузка завершена.
```

The terminal window below shows the upload process:

```
Скетч использует 1172589 байт (89%) памяти устройства. Всего доступно 1310720 байт.
Глобальные переменные используют 43080 байт (13%) динамической памяти, оставляя 284600 байт
esptool.py v2.6
Serial port COM6
Connecting....._
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC: 84:cc:a8:4d:3a:b0
<           >
115
ESP32 Dev Module на COM6
```

Рис. 3.4. Процесс начала загрузки кода в модуль беспроводной связи

После окончания загрузки в терминал будет выведена информация о том, что загрузка завершена, и появится сообщение о необходимости аппаратной перезагрузки (Рис. 3.5.).

The terminal window shows the completion of the code upload:

```
Загрузка завершена.

Writing at 0x00008000... (100 %)
Wrote 3072 bytes (144 compressed) at 0x00008000 in 0.0 seconds (effective 2730.7 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

Рис. 3.5. Успешное завершение загрузки кода

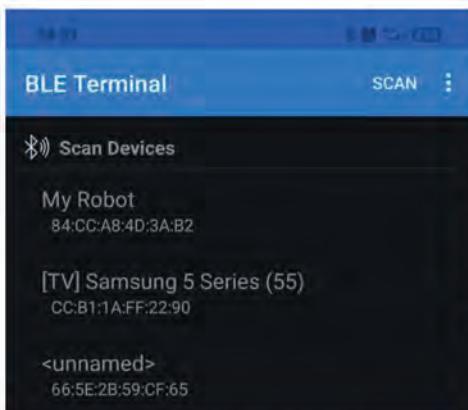


Рис. 3.6. Устройство «My Robot» в списке доступных Bluetooth - устройств

Для получения данных микроконтроллером, необходимо реализовать следующий скетч и загрузить его в Atmega2560 обычным способом - как в обычную плату Mega2560:

```
char c;
void setup() {
    // Инициализируем взаимодействие по последовательному порту с ПК
    Serial.begin (115200);
    // Инициализируем взаимодействие по последовательному порту с модулем беспроводной связи
    Serial2.begin (115200);
}

void loop(){
while(Serial2.available()){
c=Serial2.read();
Serial.print(c);
}
delay(10);
}
```

В результате, микроконтроллер будет ожидать поступление данных и, в случае их получения, будет выводить их в монитор порта (Рис. 3.7-3.8):

Для выполнения аппаратной перезагрузки необходимо отключить модуль от питания, после чего снова подключить его. После этого в окружении появится Bluetooth - устройство «My Robot» (Рис. 3.6). После выполнения настройки модуля беспроводной связи его можно переключить на работу с микроконтроллером. Для этого необходимо переместить переключатель на контроллере в положение MCU.

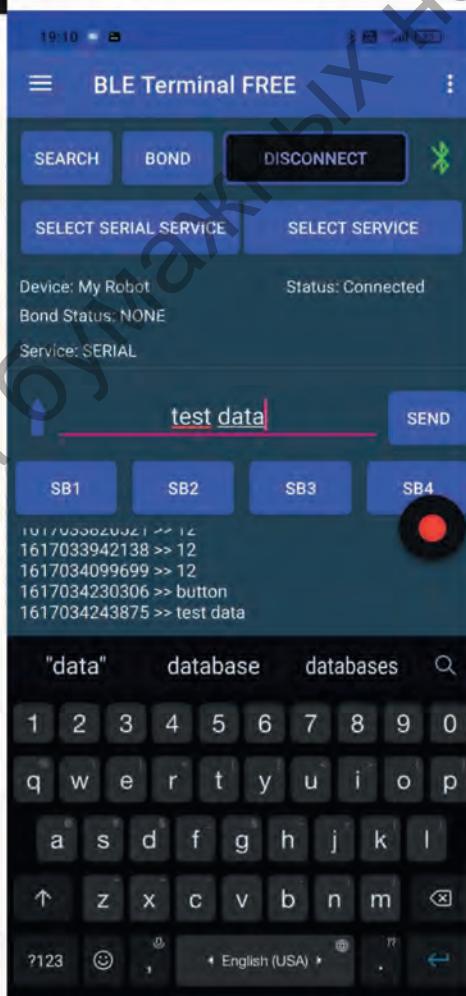


Рис. 3.7. Отправка данных из приложения

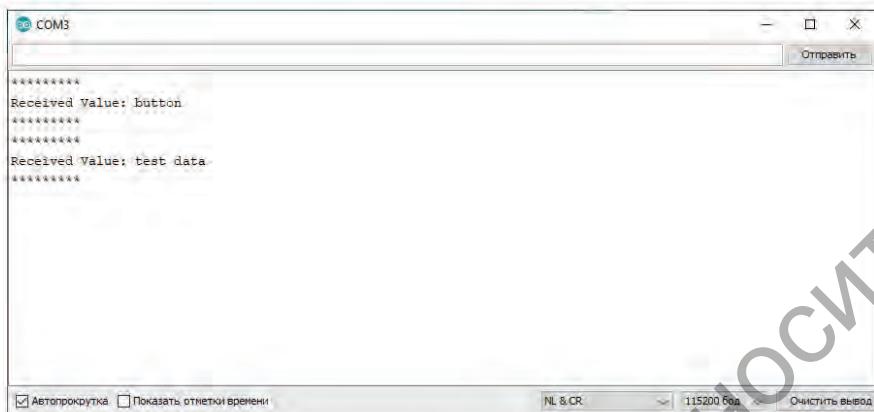


Рис. 3.8. Получение данных контроллером

Таким образом, в зависимости от поступающих данных, можно реализовывать ветвление программы - например, использовать входящие данные для управления мобильной платформой. В таком случае, на мобильном устройстве (телефоне или планшете) необходимо выбрать пульт управления, поддерживающий обмен данными по интерфейсу BLE, посмотреть какие данные он передает при нажатии на кнопки и отслеживать их в управляющей программе.

3.4. Использование WiFi-адаптера

3.4.1. Работа в качестве WiFi клиента

В данном режиме работы модулю требуется наличие внешней точки доступа и развернутой Wi-Fi сети для подключения к ней. Рассмотрим пример, в котором будет произведено подключение к существующей сети Wi-Fi и организован обмен данными между контроллером DXL-IoT и каким-либо еще устройством, подключенным к этой же Wi-Fi сети. В качестве базового примера, на основе которого будет создаваться код будет использоваться стандартный пример, по умолчанию входящий в библиотеку для работы с модулем и содержащийся в меню «Файл»->«Примеры»->«Примеры для ESP32 Dev Module»->«WiFi»->«SimpleWiFiServer».

В первую очередь необходимо подключить библиотеку WiFi.h:

```
#include <WiFi.h>
```

Затем необходимо будет задать ssid и пароль сети, к которой будет выполняться подключение:

```
const char* ssid    = «Wifi_for_Robots»;
const char* password = «1234567890»;
```

после чего выполняется инициализация сервера на 80 порту

```
WiFiServer server(80);
```

в основной функции `setup()` проинициализируем соединение по последовательному порту на скорости 115200 бод

```
Serial.begin(115200);
```

делаем небольшую паузу
`delay(10);`

и инициализируем выполнение подключения к WiFi сети с ранее заданными параметрами:

```
WiFi.begin(ssid, password);
```

после начала подключения выполняем ожидание до тех пор, пока модуль не подключится к заданной сети:

```
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(«.»);
}
```

при успешном соединении в последовательный интерфейс выводится информация об успешном соединении и выводится присвоенный модулю IP адрес:

```
Serial.println(«»);
Serial.println(«WiFi connected.»);
Serial.println(«IP address: «);
Serial.println(WiFi.localIP());
```

после инициализации сервера:
`server.begin();`

Теперь модуль готов к обмену данными. В функции `void loop()` добавим ожидание подключения клиента. И в случае подключившегося клиента, выполняем чтение данных, поступающих от него. В случае, если клиент отключается, выводится соответствующее сообщение:

```
WiFiClient client = server.available();
if (client) {
    Serial.println(«New Client.»);
```

```

String currentLine = «»;
while (client.connected()) {
    if (client.available()) {
        char c = client.read();
        Serial.write(c);
    }
}
client.stop();
Serial.println(«Client Disconnected.»);
}

```

В результате, получившийся код будет выглядеть следующим образом:

```

#include <WiFi.h>

const char* ssid    = «Wifi_for_Robots»;
const char* password = «1234567890»;

WiFiServer server(80);

void setup()
{
    Serial.begin(115200);

    delay(10);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED){
        delay(500);
        Serial.print(«.»);
    }

    Serial.println(«»);
    Serial.println(«WiFi connected.»);
    Serial.println(«IP address: »);
    Serial.println(WiFi.localIP());

    server.begin();
}

int value = 0;

void loop()
{
    WiFiClient client = server.available();

    if (client){

```

```
Serial.println("New Client.");
String currentLine = "";
while (client.connected()) {
    while (client.available()) {
        char c = client.read();
        Serial.write(c);
    }
}
client.stop();
Serial.println("Client Disconnected.");
}

}
```

Таким образом, код для работы модуля в режиме WiFi готов, и его можно загрузить стандартным способом, рассмотренным ранее.

В качестве примера получения данных в микроконтроллер загрузим такой же код, как тот, что мы использовали в предыдущем примере. Перед обменом данными с контроллером необходимо узнать, какой ему присваивается IP адрес в сети. Данный адрес может быть не постоянным, поэтому необходимо периодически его проверять. Для этого необходимо, загрузив программный код в модуль, открыть терминал и дождаться подключения его к сети – в случае успешного подключения будет выведен присвоенный модулю IP адрес (Рис. 3.9).

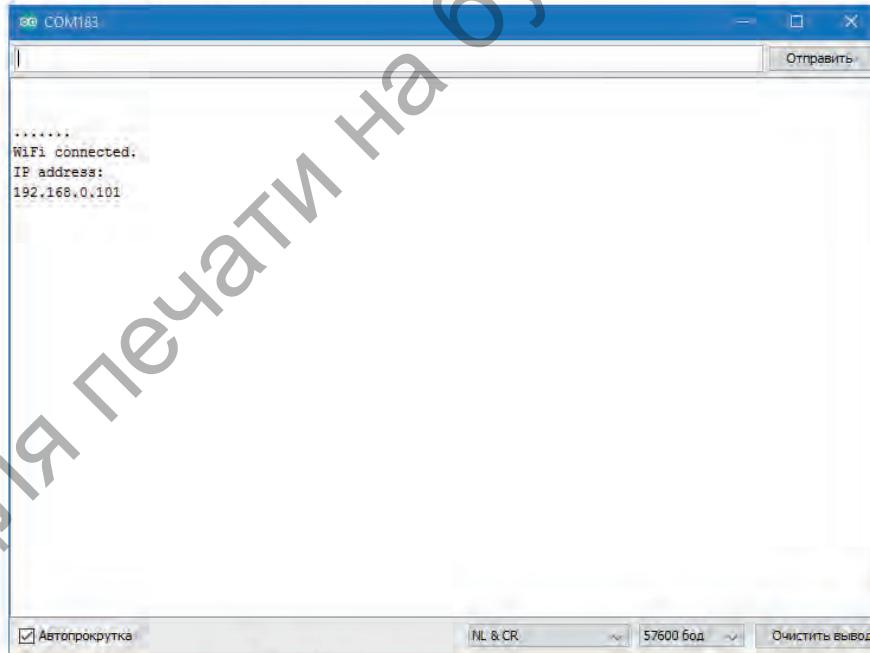


Рис. 3.9. Инициализация модуля в сети

Таким образом, используя выданный IP адрес, можно выполнять обмен данными между модулем и удаленным устройством.

Для обмена данными можно воспользоваться приложением TCP Client, где требуется произвести настройку сервера (на рисунке 3.10 это testsq) и подключиться к нему.

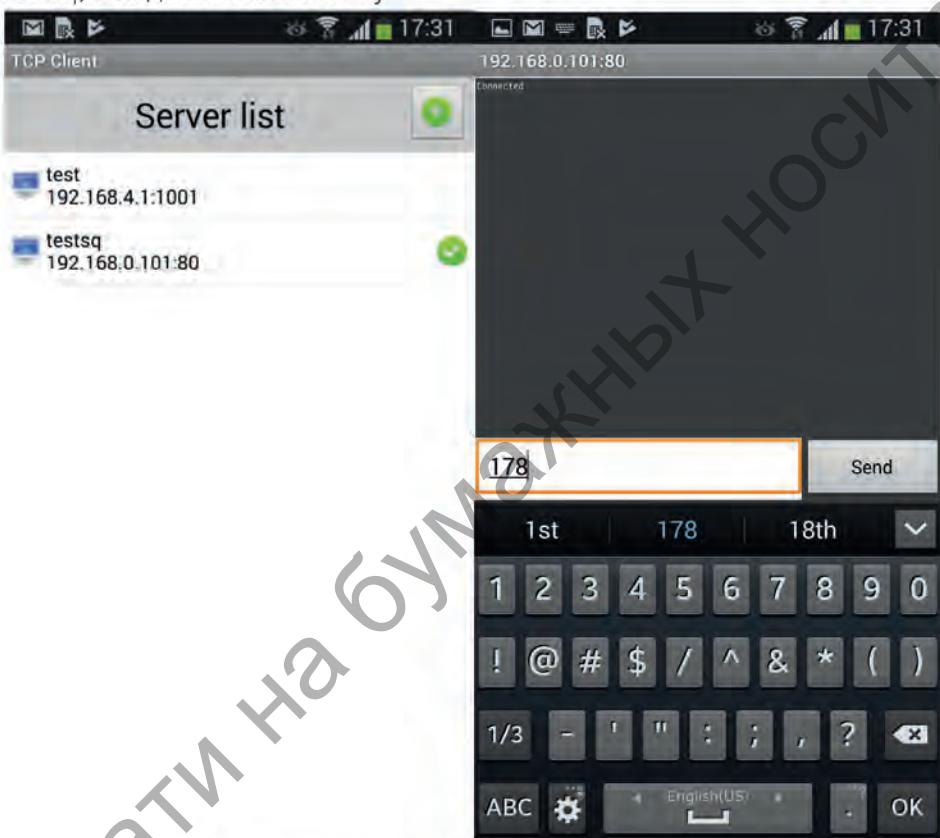


Рис. 3.10. Подключение к TCP серверу и отправка ему данных

В случае успешного подключения, откроется окно с терминалом, в котором можно отправлять и получать данные от удаленного модуля. Отправив значение «178», оно будет выведено в монитор порта.

Для передачи данных со стороны контроллера необходимо внести изменения в прошивку контроллера таким образом, чтобы данные передавались, например, каждую секунду:

```

char c;
void setup() {
    // Инициализируем взаимодействие по последовательному порту с ПК
    Serial.begin(115200);
    // Инициализируем взаимодействие по последовательному порту с модулем беспроводной связи
    Serial2.begin(115200);
}

void loop() {
    while(Serial2.available()){
        c=Serial2.read();
        Serial.print(c);
    }
    delay(1000);
    Serial2.println(<<2345>>);
}

```

Загрузив данный код в контроллер и подключившись в сеть в окне терминала TCP Client отобразятся передаваемые данные (Рис. 3.11).

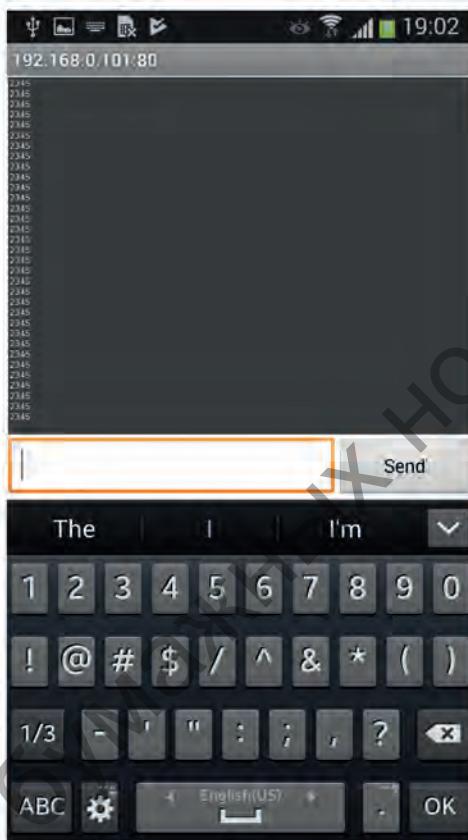


Рис. 3.11. Прием данных со стороны контроллера

На основе рассмотренных примеров, при помощи модуля беспроводной связи контроллера можно организовать обмен данными между устройствами по интерфейсу WiFi. При необходимости, также можно выводить данные в окно браузера, передавать в облачные приложения и реализовывать удаленные системы контроля и управления. Для освоения таких приемов рекомендуется изучить имеющиеся стандартные примеры.

3.4.2. Работа в качестве WiFi точки доступа

При необходимости, модуль беспроводной связи может сам стать источником WiFi сети. Для этого необходимо внести следующие изменения в функцию `setup()` написанной ранее программы, после которых она должна принять следующий вид:

```

Serial.begin(115200);

WiFi.softAP(ssid, password);

Serial.println();
Serial.print(<<IP address: >>);
Serial.println(WiFi.softAPIP());

```

при этом можно изменить наименование сети:

```
const char* ssid = «Wifi from BT-AR-01»;
```

Загрузив измененный код в программу, можно убедиться в создании новой сети, проверив IP адрес способом, рассмотренным ранее.

Используя вышеописанный метод, была создана новая Wi-Fi сеть «*WiFi from BT-AR-01*», к которой можно подключиться и выполнить все те же задачи, которые были рассмотрены ранее (Рис. 3.12).

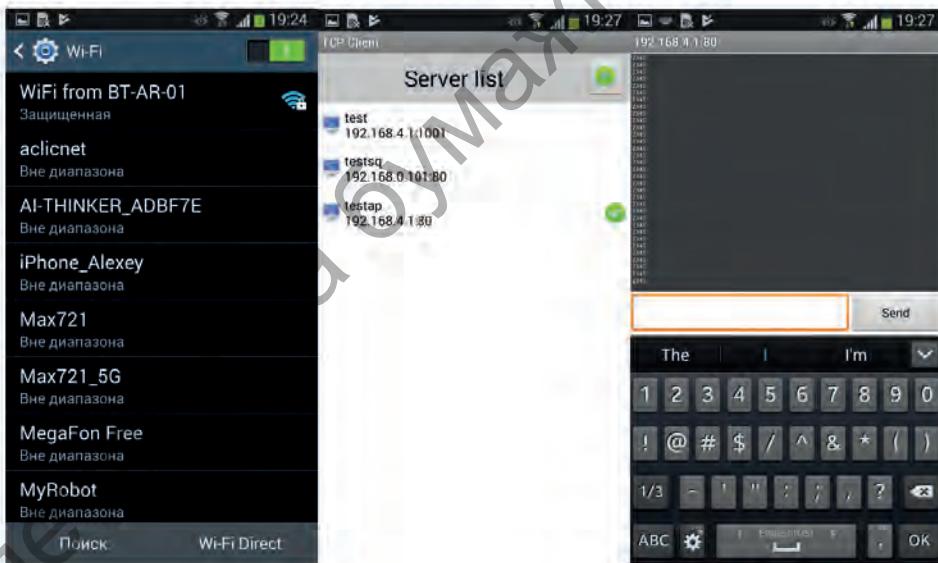


Рис. 3.12. Работа в качестве точки доступа

Таким образом, модуль беспроводной связи, встроенный в программируемый контроллер DXL-IoT может быть использован и как клиент WiFi сети, и как ее источник, что дает широкие возможности при организации обмена данными между программируемым контроллером и произвольным удаленным устройством.

3.5. Использование платы расширения с адаптером Ethernet

Для подключения универсального вычислительного контроллера DXL-IoT в сеть проводным образом существует плата расширения с адаптером Ethernet, позволяющая подключить контроллер в сеть на скорости до 100 Мб/сек. Помимо этого, на модуле находится слот для установки microSD карты памяти, которую можно использовать как хранилище файлов. Для программной реализации Ethernet-соединения в среде разработки Arduino IDE существует набор готовых примеров во вкладке «Файл» -> «Примеры» -> «Ethernet».

В качестве демонстрационного примера можно рассмотреть пример передачи данных в браузер персонального компьютера. В случае подключения контроллера напрямую к компьютеру через Ethernet, либо же при включении его в сеть, где все устройства имеют статический IP, необходимо указать контроллеру такой IP, чтобы он попадал в подсеть сети и компьютера. Также для работы с чипом необходимо инициализировать следующие требующиеся для этой задачи библиотеки:

```
#include <SPI.h> // Подключаем библиотеку SPI
#include <Ethernet.h> // Подключаем библиотеку Ethernet

byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED}; // Вводим MAC адрес
IPAddress ip(169, 254, 21, 70); // Указываем статический IP

EthernetServer server(80); // Инициализация библиотеки
Ethernet «server» на порту 80
```

Затем задаем скорость передачи данных и ждем, пока не откроется монитор порта, после чего запускаем сервер:

```
void setup()
{
    Serial.begin(115200); // Задаем скорость передачи
    // данных
    while (!Serial) {}
    Ethernet.begin(mac, ip); // Запускаем сервер
    server.begin();
    Serial.print("server is at ");
    Serial.println(Ethernet.localIP());
}
```

В функции `loop()` ожидаем подключение клиента (открытие вкладки браузера с указанием в адресной строке `IP:port`). Начинается передача данных, которые в данном примере представлены значениями с аналоговых портов:

```

void loop()
{EthernetClient client = server.available(); // Принимаем данные, посы-
ляемые клиентом
if (client){
  Serial.println("new client");
  boolean currentLineIsBlank = true;
  while (client.connected()){
    if (client.available()){
      char c = client.read();
      Serial.write(c);
      if (c == '\n' && currentLineIsBlank){
        client.println("HTTP/1.1 200 OK");
        client.println("Content-Type: text/html");
        client.println("Connection: close"); // the connection will be closed
        after completion of the response
        client.println("Refresh: 5"); // refresh the page automatically every
5 sec
        client.println();
        client.println("<!DOCTYPE HTML>");
        client.println("<html>");
        for (int analogChannel = 10; analogChannel < 16; analogChannel++)
{int sensorReading = analogRead(analogChannel);
  client.print("analog input ");
  client.print(analogChannel);
  client.print(" is ");
  client.print(sensorReading);
  client.println("<br />");}
        client.println("</html>");
        break;}
      if (c == '\n'){
        currentLineIsBlank = true;
      } else if (c != '\r'){
        currentLineIsBlank = false;
      }
    }
    delay(1);
    client.stop();
    Serial.println("client disconnected");
  }
}

```

В результате, введя IP адрес сервера и порт в браузерной строке, откроется окно, в котором появятся данные, отправляемые контроллером.

3.6. Использование силовой платы расширения

Силовая плата расширения для программируемого контроллера DXL-IoT предназначена для управления силовой нагрузки. Такой нагрузкой могут выступать популярные двигатели постоянного тока, светодиодные ленты, хоббийные пневмо- и гидро- насосы. Данная плата может одновременно управлять двигателями постоянного тока (задавая направление и скорость вращения), в количестве до четырех штук, а также коммутиировать питание до четырех устройств. Здесь необходимо понимать, что чем больше устройств будет подключено к плате расширения, тем больше линий будет зарезервировано для них, и тем меньше линий на штыревых разъемах будут доступны пользователю для работы. Внешний вид типовой схемы подключения двух двигателей постоянного тока к плате показан на рисунке 3.13:

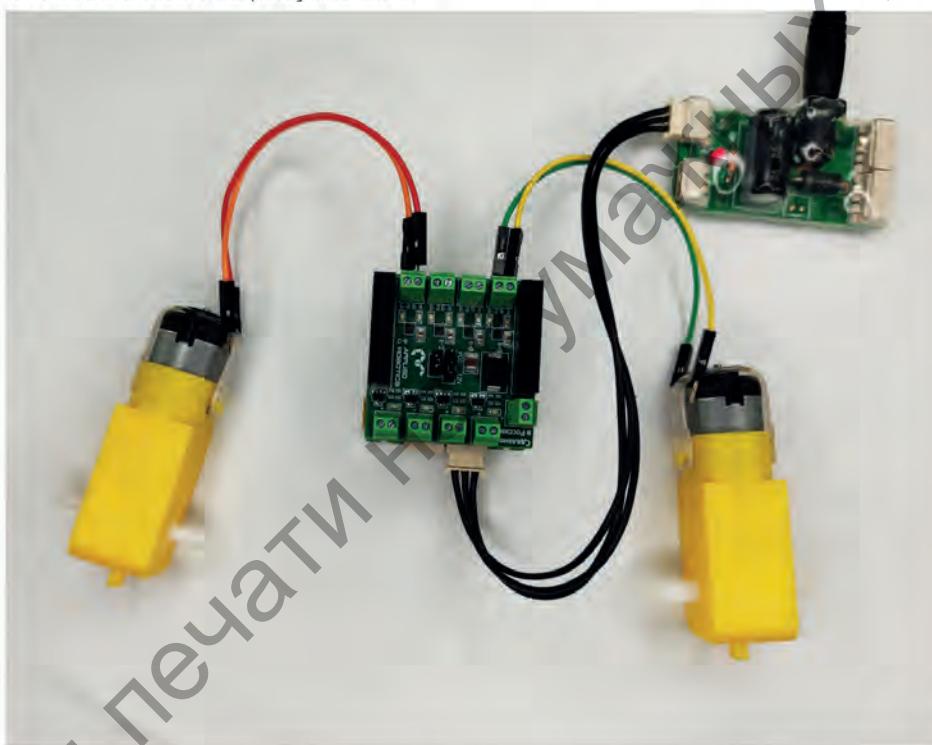


Рис. 3.13. Подключение двигателей постоянного тока к силовой плате расширения

В данном примере контроллер питается от источника 12В. И джамперы на плате расширения стоят таким образом, что эти 12В пойдут на питание двигателей.

Пример управления двигателем постоянного тока подключенного к клемнику 5 выглядит следующим образом:

```
#define M1_dir 2
#define M1_Speed 3

volatile int val = 50;

void setup(){
pinMode(M1_dir, OUTPUT);
pinMode(M1_Speed, OUTPUT);

digitalWrite(M1_dir, LOW);
analogWrite(M1_Speed, val);

delay(2000);

digitalWrite(M1_dir, HIGH);
analogWrite(M1_Speed, val);

delay(2000);
val=0;
analogWrite(M1_Speed, val);

}

void loop(){

}
```

Важно! В последних версиях Arduino IDE существует баг, не позволяющий корректно отрабатывать функции analogWrite. Для того, чтобы избежать этой ошибки, необходимо значение скорости писать не напрямую, как аргумент функции, а передавать в функцию через переменную int с квалификатором volatile.

Аналогичным образом управляются все остальные каналы для подключения двигателей постоянного тока.

Для подачи силового питания на клемники 1-4 можно воспользоваться следующим примером:

```
#define K1 10
#define K2 11
#define K3 12
#define K4 13

void setup(){
pinMode(K1, OUTPUT);
```

```
pinMode(K2, OUTPUT);
pinMode(K3, OUTPUT);
pinMode(K4, OUTPUT);

digitalWrite(K1, HIGH);
delay(2000);
digitalWrite(K1, LOW);

digitalWrite(K2, HIGH);
delay(2000);
digitalWrite(K2, LOW);

digitalWrite(K3, HIGH);
delay(2000);
digitalWrite(K3, LOW);

digitalWrite(K4, HIGH);
delay(2000);
digitalWrite(K4, LOW);

}

void loop(){

}
```

Здесь последовательно активируются клемники с 1 по 4, и через две секунды деактивируются. Индикационный светодиод на плате показывает состояние клемника.

3.7. Управление Dynamixel-совместимыми устройствами

3.7.1. Управление сервоприводами DYNAMIXEL

Одной из особенностей программируемого контроллера DXL-IoT является возможность подключения и управления сервоприводами DYNAMIXEL. Для программной реализации системы управления понадобится библиотека DxIMaster, рассмотренная ранее. В качестве примера рассмотрим схему подключения и процесс управления сервоприводом AX-12A.

Поскольку сервопривод является устройством, которое потребляет достаточное количество тока и рекомендуется его питать напряжением 12В, то необходимо использовать внешний источник питания. Схема подключения устройств выглядит так: внешний источник питания подключается к плате распределения питания (SMPS2Dynamixel), и уже к ней подключается контроллер DXL-IoT и сервопривод (Рис. 3.14)



Рис. 3.14. Подключение сервопривода AX-12A к контроллеру DXL-IoT

Таким образом, питание 12В раздается на все устройства, подключенные в цепь от платы распределения питания. При этом, программируемый контроллер также можно подключить к ПК с помощью USB кабеля. Если требуется подключение дополнительных устройств, то их можно подключать как к сервоприводу, так и к контроллеру в свободные 3х pin-новые порты.

Для управления сервоприводом реализуем следующий скетч на основе примера joint_mode из раздела «Файл» -> «Примеры» -> «Примеры из пользовательских библиотек» -> «DxlMaster» -> «joint_mode»:

```
#include <<DxlMaster.h>>

// id of the motor
const uint8_t id = 1;
// speed, between 0 and 1023
int16_t speed = 512;
// communication baudrate
const long unsigned int baudrate = 1000000;

DynamixelMotor motor(id);

void setup()
{
    DxlMaster.begin(baudrate);
    delay(100);

    // check if we can communicate with the motor
    // if not, we stop here
    uint8_t status = motor.init();
    if(status != DYN_STATUS_OK)
    {
        while(1);
    }

    motor.enableTorque();

    // set to joint mode, with a 300° angle range
    // see doc to compute angle values
    motor jointMode(0, 1024);
    motor.speed(speed);
}

void loop()
{
    // go to middle position
    motor.goalPosition(512);
    delay(500);

    // move 45° CCW
```

```

motor.goalPosition(666);
delay(500);

// go to middle position
motor.goalPosition(512);
delay(500);

// move 45° CW
motor.goalPosition(358);
delay(500);
}

```

В результате загрузки данного кода в контроллер, подключенный к нему сервопривод, имеющий ID = 1 и скорость обмена данными в 1000000 бод, будет перемещать свой фланец в диапазоне от -45 до 45 градусов от центральной позиции. Таким образом можно управлять положением фланца сервопривода. В аналогичном примере (wheel_mode) показан способ управления приводом в режиме непрерывного вращения.

3.7.2. Управление Dynamixel-совместимыми периферийными модулями

Помимо сервоприводов DYNAMIXEL с помощью универсального вычислительного контроллера DXL-IoT можно управлять различными периферийными модулями, обладающими интерфейсом Dynamixel. Такие модули производятся, например, компанией «Прикладная робототехника» и о них более подробно рассказывается в соответствующем учебном пособии. В данном случае, в качестве примера, рассмотрим процесс управления трехцветным светодиодом (Рис. 3.15).

```

#include «DxlMaster.h»

const unsigned long dynamixel_baudrate = 1000000;
const unsigned long serial_baudrate = 57600;
const uint8_t id = 21;

DynamixelDevice rgb(id);
void setup(){
  DxIMaster.begin(dynamixel_baudrate);
  rgb.init();
  Serial.begin(serial_baudrate);
}

void loop(){
  rgb.write(26, 255);
}

```



Рис. 3.15. Подключение периферийного модуля к контроллеру DXL-IoT

```
delay(1000);
rgb.write(27, 255);
delay(1000);
rgb.write(28, 255);
delay(1000);
rgb.write(26, 0);
delay(1000);
rgb.write(27, 0);
delay(1000);
rgb.write(28, 0);
delay(1000);
}
```

В данном примере модуль трехцветного светодиода, имеющего ID = 0x15 (21 в десятичной системе) и настроенного на скорость обмена данными в 1000000 бод, управляется путем последовательного включения и выключения всех цветовых составляющих.

3.7.3. Опрос Dynamixel-совместимого периферийного модуля

Опрос Dynamixel-совместимых периферийных модулей выполняется образом, аналогичным управлению такими модулями. У каждого модуля существует регистры, в которых содержится значимая информация. И весь опрос сводится к тому, что этот регистр необходимо опросить. Рассмотрим пример опроса кнопки, при нажатии на которую будет включаться светодиод на модуле трехцветного светодиода (Рис. 3.16):



Рис. 3.16. Одновременное подключение нескольких периферийных модулей к контроллеру DXL-IoT

```
#include «DxlMaster.h»  
  
const unsigned long dynamixel_baudrate = 1000000;  
const unsigned long serial_baudrate = 57600;  
const uint8_t idRGB = 21;  
const uint8_t idBUT = 3;  
uint8_t a;  
  
DynamixelDevice rgb(idRGB);  
DynamixelDevice but(idBUT);  
  
void setup(){  
    DxlMaster.begin(dynamixel_baudrate);  
    rgb.init();  
    but.init();  
    Serial.begin(serial_baudrate);  
}
```

```
void loop(){
    but.read(27, a);
    if (a==1){
        rgb.write(26, 255);}
    else rgb.write(26, 0);
}
```

В результате, при нажатии на кнопку, на модуле трехцветного светоизлучающего диода будет включаться зеленый свет.

3.8. Конфигурирование контроллера, как Dynamixel-совместимое устройство

Универсальный вычислительный контроллер DXL-IoT, помимо того, что может управлять Dynamixel-совместимыми устройствами, в свою очередь, может быть сам сконфигурирован как Dynamixel-совместимое устройство. Тем самым, давая возможность расширять функционал других контроллеров или создавать на своей основе специфические периферийные модули, которые можно интегрировать в цепочку Dynamixel-совместимых устройств.

Для конфигурирования необходимо использовать библиотеку DxISlave, устанавливаемую в среду разработки аналогично библиотеке DxIMaster. Процесс эмуляции сводится к установке в таблицу регистров необходимых значений, корректно воспринимаемых системой. Рассмотрим пример эмуляции устройства со следующими параметрами:

```
device model 1234
software version 12
dxl id 1
baud rate 1 (1000000 bps)
```

Данный пример расписан в «Файл» -> «Примеры» -> «Примеры из пользовательских библиотек» -> «DxISlave» -> «allRegs»

Исходя из резервирования ячеек таблицы регистров под системные нужды, необходимо первым делом корректно сконфигурировать device model, software version, dxl id и baud rate:

```
DxISlave.begin(1234, 12);
Serial.begin(9600);

// init EEPROM only on first boot
if(!DxISlave.get(IS_INITED_REG))
{
```

```

Serial.println("First boot");
DxISlave.set(IS_INITED_REG, 1);

DxISlave.set_id(1); // the same as DxISlave.set(3, 1);
DxISlave.set_baud(1); // the same as DxISlave.set(4, 1);

// fill regs with values = addr
for(int i = 6; i < 23; i++)
    DxISlave.set(i, i);
}

```

Причем, при первой конфигурации устройства необходимо убедиться, что параметры были корректно записаны в EEPROM (проверяется состояние флага в регистре IS_INITED_REG).

После инициализации основных регистров можно переходить к заполнению произвольных. Например, далее регистры до 200 заполняются значениями своих адресов, и при этом адресам выше 100 ставится режим «Только для чтения».

```

for(int i = 24; i < 200; i++)
{
    DxISlave.set(i, i);

    // set read only access modes to some regs
    if(i >= 100)
        DxISlave.set_mode(i, 0);
}

```

В качестве примера показан способ заполнения регистров 16-битными значениями (одной командой заполняется сразу 2 регистра - текущий, к которому выполняется обращение, и следующий):

```

// fill some regs as 16 bit pairs
for(int i = 200; i < 255; i += 2)
{
    DxISlave.set16(i, 1000 + i);

    // set read only mode for some reg pairs
    if(i >= 240)
        DxISlave.set_mode16(i, 0);
}

```

Модифицируем имеющийся пример allRegs таким образом, чтобы в один из регистров, например, 51, будут последовательно записываться

значения от 0 до 255 с периодом в 0.5 с.

```
#define IS_INITED_REG 23

uint32_t cur_time = millis();

void setup(){
    // put your setup code here, to run once:
    // init dxl slave with model number 1234 and software version 12
    DxlSlave.begin(1234, 12);
    Serial.begin(9600);

    // init EEPROM only on first boot
    if(!DxlSlave.get(IS_INITED_REG))
    {
        Serial.println("First boot");
        DxlSlave.set(IS_INITED_REG, 1);

        DxlSlave.set_id(1); // the same as DxlSlave.set(3, 1);
        DxlSlave.set_baud(1); // the same as DxlSlave.set(4, 1);

        // fill regs with values = addr
        for(int i = 6; i < 23; i++)
            DxlSlave.set(i, i);
    }
}

void loop(){
    // fill RAM regs
    for(int i = 0; i < 255; i++)
    {
        DxlSlave.set(51, i);
        delay(500);
    }
}
```

Считаем это значение через контроллер СМ-530. В результате, в терминал будут выведены значения, содержащиеся в регистре 51 (Рис. 3.17):

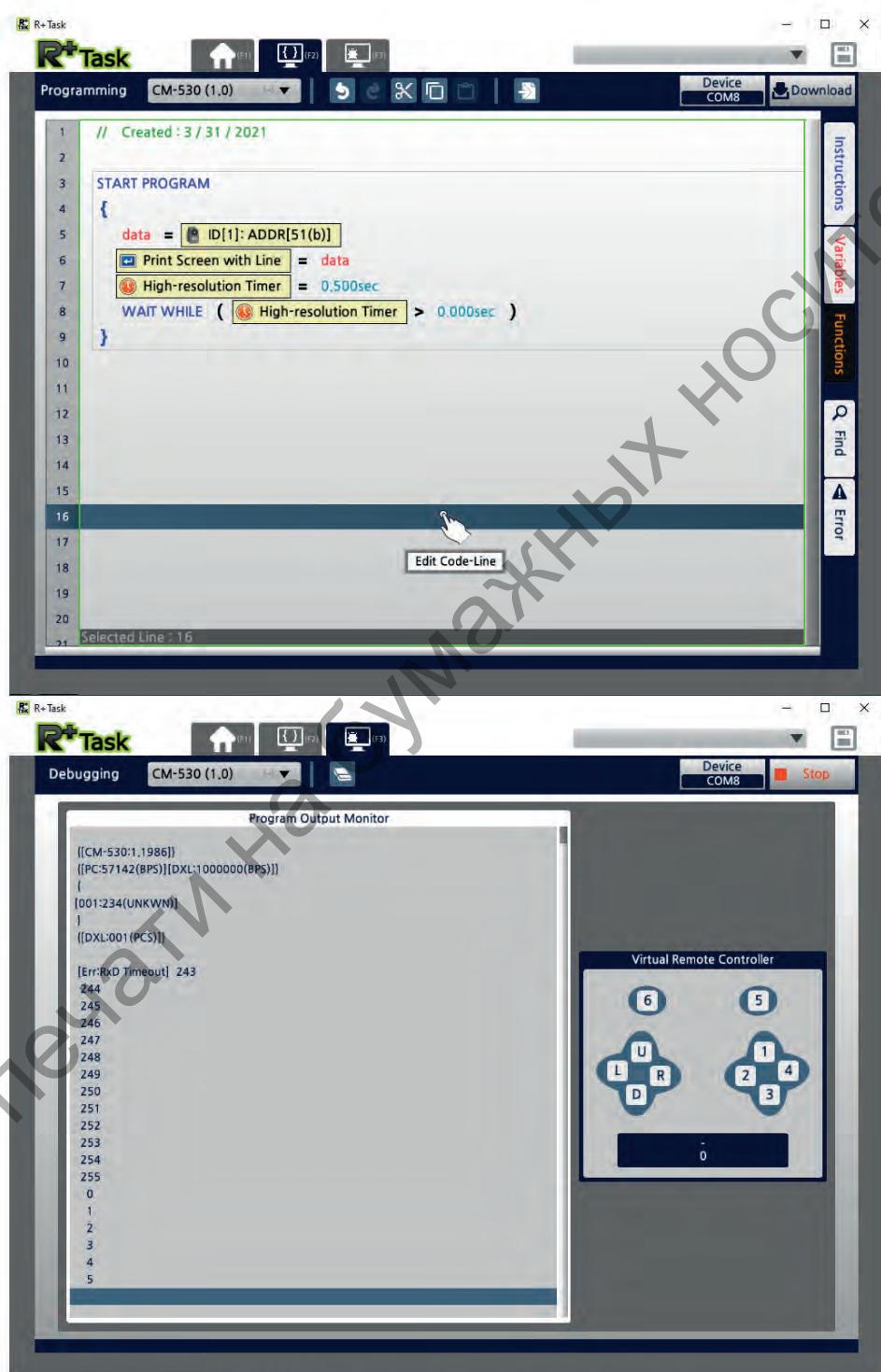


Рис. 3.17. Чтение данных с контроллера DXL-IoT

3.9. Управление мобильной платформой через Web-интерфейс

В качестве финального примера рассмотрим пример управления мобильной платформой дифференциального типа, используя сетевое соединение и веб-интерфейс.

В начале рассмотрим, как работает контроллер DXL-IoT с интернет-архитектурой. В среде Arduino IDE имеется пример по развертыванию веб-сервера на базе контроллера Arduino. Для того, чтобы проверить работу нашего контроллера, необходимо подключить Ethernet кабель к роутеру и программируемому контроллеру, используя для этого Ethernet-модуль расширения. На контроллер потребуется загрузить скетч *WebService* из примера *Ethernet*. В этом скетче можно найти поле, где указывается IP адрес веб-страницы.

```
20 #include <SPI.h>
21 #include <Ethernet.h>
22
23 // Enter a MAC address and IP address for your controller below,
24 // The IP address will be dependent on your local network:
25 byte mac[] = {
26   0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
27 };
28 IPAddress ip(192, 168, 1, 177);
29
30 // Initialize the Ethernet server library
31 // with the IP address and port you want to use
32 // (port 80 is default for HTTP):
33 EthernetServer server(80);
```

Важно, чтобы ПК был подключен к той же сети, что и программируемый контроллер, а также, чтобы IP адрес компьютера и контроллера были в одной подсети. Подключив ПК к той же сети и перейдя по указанному IP адресу, можно увидеть веб-страницу, развернутую на контроллере.

В данную веб-страницу контроллер передает данные с аналоговых датчиков.

В качестве объекта управления будет рассматриваться мобильная платформа дифференциального типа с установленными на ней сервоприводами DYNAMIXEL. Управление данной платформой будет осуществляться с помощью веб-интерфейса, генерируемого на стороне контроллера и доступного в окне браузера любого устройства, подключенного к сети с платформой.

Начнем с инициализации системы:

```

2 #include "SPI.h"
3 #include "Ethernet.h"
4 #include "DxlMaster.h"
5 const uint8_t id = 1;
6 const uint8_t id1 = 2;
7 int16_t speed = 512;
8 const long unsigned int baudrate = 1000000;
9 DynamixelMotor motor(id);
10 DynamixelMotor motor1(id1);
11 boolean newInfo = 0;
12 byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
13 IPAddress ip(192, 168, 1, 177);
14 EthernetServer server(80);

```

В этой части, как и во всех скетчах, объявляются используемые библиотеки: библиотека для работы с SPI, библиотека для работы с Ethernet, библиотека для работы с интерфейсом Dynamixel. Далее声明аются ID сервоприводов. В данном случае, ID равны 1 и 2. Также声明ается переменная, содержащая значение скорости - speed. В строке 8 задается скорость обмена сообщениями с сервоприводами, и далее создаются экземпляры класса для работы с сервоприводами. Также необходимо создать логическую переменную, которая будет использоваться для отслеживания состояния - обновилась ли информации или нет. В строке 12 задается mac адрес (обычно он такой же, и менять его не надо). Далее указывается IP адрес, на котором будет находиться наш сервер (он должен находиться в одной подсети с ПК). В строке 14 происходит инициализация сервера на заданном порту.

```

16 void setup() {
17
18 Ethernet.begin (mac, ip);
19 Serial.begin (1000000);
20 server.begin();
21 DxlMaster.begin(baudrate);
22 delay(100);
23 uint8_t status = motor.init();
24 motor.enableTorque();
25 motor.wheelMode();
26 motor1.enableTorque();
27 motor1.wheelMode();
28 }

```

В этой части скетча инициализируется и запускается сервер с указанными ранее параметрами. Так же происходит инициализация сервоприводов и задание режимов их работы.

Основная работа происходит в функции loop().

```
30 void loop() {  
31   EthernetClient client = server.available();  
32   if (client){  
33     boolean currentLineIsBlank = true;  
34  
35     while (client.connected()) {  
36       if (client.available()) {  
37         char c = client.read();  
38         if (newInfo && c == '\n'){  
39           newInfo = 0;  
40         }  
41  
42         if (c == '$'){  
43           newInfo = 1;  
44         }  
45         if (newInfo == 1){  
46           Serial.println (c);  
47           if (c == '1'){  
48             Serial.println ("Вперед");  
49             speed =100;  
50             motor.speed(speed);  
51             motor1.speed(-1 * speed);  
52           }  
53           if (c == '2'){  
54             Serial.println ("Разворот");  
55             speed =-100;  
56             motor.speed(speed);  
57             motor1.speed(speed);  
58           }  
59           if (c == '3'){  
60             Serial.println ("Стоп");  
61             speed =0;  
62             motor.speed(speed);  
63             motor1.speed(speed);  
64           }  
65         }  
}
```

В начале происходит проверка наличия соединения между клиентом и сервером (принимаются ли данные от клиента). Если соединение есть, то программа входит в цикл while и не выходит из него, пока соединение не оборвется. Далее, если клиент активен, то происходит считывание передаваемой информации и записывание ее в переменную «с». Затем происходит обработка полученной информации и праведение анализа, насколько эта информация нова. Если переменная новой информации = 1 и «с», в которой записан запрос, равен пустой строке, то обнуляется переменная поступления новой информации. Если переменная «с», несущая отправленный запрос, содержит символ \$, то пришла новая информация, ставится метка новой информации.

После проверяется содержание URL. В данном случае можно получить значения от 1 до 3. При получении значения «1», контроллер передает команду на вращение сервоприводами в разные стороны, но с одинаковой скоростью. За счет того, что сервопривода располагаются зеркально, мобильная платформа едет вперед. При получении значения «2», мобильная платформа будет совершать поворот на месте, вокруг своей оси. При получении значения «3», мобильная платформа остановится.

```
66 if (c == '\n') {
67 currentLineIsBlank = true;
68 }
69 else if (c != '\r') {
70
71 currentLineIsBlank = false;
72 }
73
74 if (c == '\n' && currentLineIsBlank) {
```

Если в переменной с записан символ новой строки, то необходимо начать новую строку. Если же переменная с не равна символу новой строки, то информация записывается в текущую строку.

```
83 client.println("HTTP/1.1 200 OK");
84 client.println("Content-Type: text/html");
85 client.println("Connection: close");
86 client.println("Refresh: 5");
87 client.println();
88 client.println("<!DOCTYPE HTML>");
89 client.println("<html>");
90 client.println("<head>");
91 client.println("<meta http-equiv='Content-Type' content='text/html ; charset=utf-8' /> ");
92 client.print("<title>IoT Web Server</title>");
93 client.println("</head>");
94 client.println("<body>");
95 client.print("<H1>IoT Web Server</H1>");
96 client.print("<a href=\"$1\"><button>Прямо</button></a>"); 
97 client.print("<a href=\"$2\"><button>Разворот</button></a>"); 
98 client.print("<a href=\"$3\"><button>Стоп</button></a>"); 
99
00 client.println("</body>");
01 client.println("</html>");
02 break;
03 }
04 }
05 }
```

Вывод страницы html выглядит следующим образом:

Здесь:

Строки 83 – 85: заголовочная информация веб-страницы. Стока 86 будет обновлять страницу каждые 5 секунд. В строках 87 – 91 указывается тип документа (html), происходит открытие тега html и Head. Далее идет название страницы, заголовочная информация и заголовок на самой странице. В строках 96 – 98 создаются кнопки «Прямо», «Разворот» и «Стоп». После чего происходит закрытие тегов и выход.

```
106 delay(1);
107 client.stop();
108 }
109 }
```

В строке 106 – реализована задержка по времени на получение новых данных с веб-страницы, а в 107 – закрытие соединения.

После того как данный скетч загружен в программируемый контроллер, а сам контроллер подключен к сети, можно зайти на созданную веб-страницу и увидеть примерно следующее:



IoT Web Server

В результате, платформа будет выполнять определенное движение при нажатии на кнопки. Полный код прошивки выглядит следующим образом:

```
include <>SPI.h>
#include <>Ethernet.h>
#include <>DxlMaster.h>
const uint8_t id = 1;
const uint8_t id1 = 2;
int16_t speed = 512;
const long unsigned int baudrate
= 1000000;
DynamixelMotor motor(id);
DynamixelMotor motor1(id1);
boolean newInfo = 0;
byte mac[] = { 0xDE, 0xAD, 0xBE,
0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 177);
EthernetServer server(80);
void setup(){
Ethernet.begin (mac, ip);
Serial.begin (1000000);
server.begin();
DxlMaster.begin(baudrate);
delay(100);
uint8_t status = motor.init();
motor.enableTorque();
```

```
motor.wheelMode();
motor1.enableTorque();
motor1.wheelMode();
}
void loop(){
EthernetClient client = server.available();
if (client){
boolean currentLineIsBlank = true;
while (client.connected()){
if (client.available()){
char c = client.read();
if (newInfo && c == '\n'
newInfo = 0;
}
if (c == '$'){
newInfo = 1;
}
if (newInfo == 1){
Serial.println (c);
if (c == '1'){
Serial.println («Вперед»);
```

```
speed =100;
motor.speed(speed);
motor1.speed(-1 * speed);
}
if (c == '2'){
Serial.println («Разворот»);
speed =-100;
motor.speed(speed);
motor1.speed(speed);
}
if (c == '3'){
Serial.println («Стоп»);
speed =0;
motor.speed(speed);
motor1.speed(speed);
}
}
if (c == '\n'){
currentLineIsBlank = true;
}
else if (c != '\r'){
currentLineIsBlank = false;
}
if (c == '\n' && currentLineIsBlank)
{
    client. println («HTTP/1.1 200
OK»);
    client. println («Content-Type:
text/html»);
    client. println («Connection:
close»);
    client. println («Refresh: 5»);
    client. println ();
    client. println («<!DOCTYPE
HTML»);
    client. println («<html>»);
    client. println («<head> »);
    client. println («<meta http-
equiv='Content-Type' content='text/
html ; charset=utf-8'/> »);
    client. print («<title>iOT Web
Server</title>»);
    client. println («</head>»);
    client. println («<body>»);
    client. print («<H1>iOT Web
```

не для печати на бумажных носителях

YOUTUBE КАНАЛ



ИНСТРУКЦИИ



ПРОГРАММНОЕ
ОБЕСПЕЧЕНИЕ



ЗД МОДЕЛИ



МОСКВА 2021