Лабораторная работа № 6

Изучение I2C

Цель лабораторной работы: познакомиться с принципами организации I2C, а также научиться использовать I2C на платформе Arduino.

Теоретическая часть

Последовательный протокол обмена данными IIC (также называемый I2C – Inter-Integrated Circuits, межмикросхемное соединение) использует для передачи данных две двунаправленные линии связи, которые называются шина последовательных данных **SDA (Serial Data)** и шина тактирования **SCL (Serial Clock)**. Также имеются две линии для питания. Шины SDA и SCL подтягиваются к шине питания через резисторы.

В сети (Рисунок 1) есть хотя бы одно **ведущее устройство (Master)**, которое инициализирует передачу данных и генерирует сигналы синхронизации. В сети также есть **ведомые устройства (Slave)**, которые передают данные по запросу ведущего. У каждого ведомого устройства есть уникальный адрес, по которому ведущий и обращается к нему. Адрес устройства указывается в паспорте (datasheet). К одной шине I2C может быть подключено до 127 устройств, в том числе несколько ведущих. К шине можно подключать устройства в процессе работы, т.е. она поддерживает «горячее подключение».

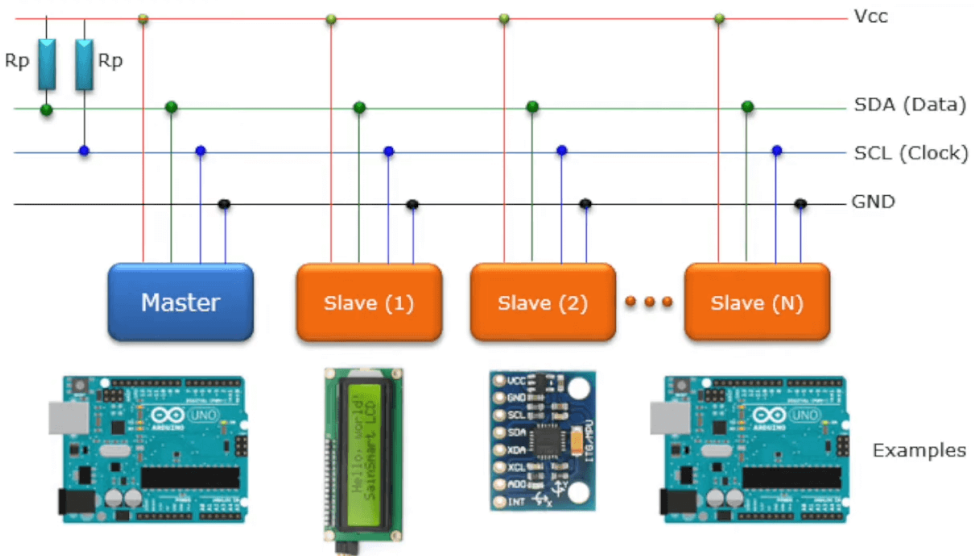

Рисунок 1 – Пример сети I2C

Шина в I2C - это просто два провода, которые соединяют все I2C-устройства в сети.Эти два провода называются SDA и SCL. Провод SDA используется для связи между ведущим и ведомым устройствами.

Линия SCL несет тактовый сигнал, используемый для правильной синхронизации связи. Для поддержания обоих проводов в состоянии HIGH необходимы подтягивающие (pull-up) резисторы.

Будьте внимательны при подключении I2C устройств к Arduino. Arduino выводит I2C-сигналы на **5В** логическом уровне, но I2C-устройства работают с различными напряжениями логического уровня.

Таким образом, I2C устройство, которое работает на **3,3 В** может быть повреждено при подключении к Arduino. В паспорте устройства должно быть указано напряжение логического уровня.

Если подтягивающие резисторы подключены к **+5В**, все устройства должны быть совместимы для работы с логическим уровнем **+5В**.

Arduino имеет специальные контакты для I2C, которые имеют встроенные подтягивающие резисторы в соответствии с требованиями протокола I2C.

Для плат Arduino Uno это контакты A4 и A5. Пин A4 - это контакт SDA, а пин A5 - это контакт SCL. В версии Arduino Uno R3 есть еще один набор контактов I2C рядом с USB-разъемом:

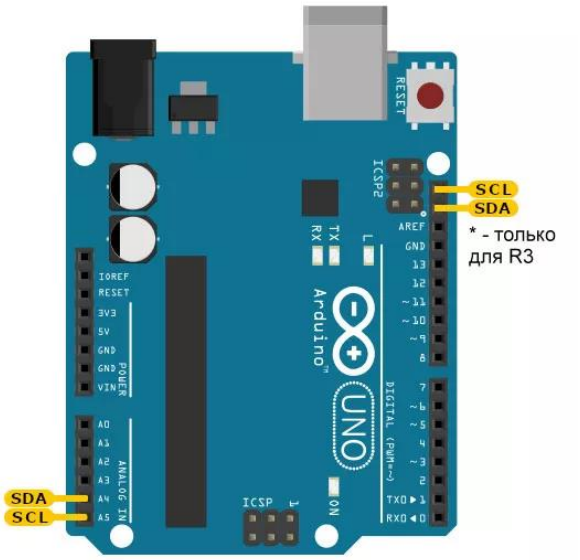

Рисунок 2 - I2C пины Arduino UNO

Чтобы продемонстрировать, как использовать I2C в Arduino, создадим проект, который посылает данные туда и обратно между двумя платформами.

Используем I2C связь для изменения скорости мерцания светодиода на пине 13 на одной Arduino, в зависимости от положения потенциометра, подключенного к другой Arduino.

Одна Arduino будет выступать в качестве ведущего (master), а другая Arduino будет выступать в качестве ведомого (slave) (Рисунок 3).

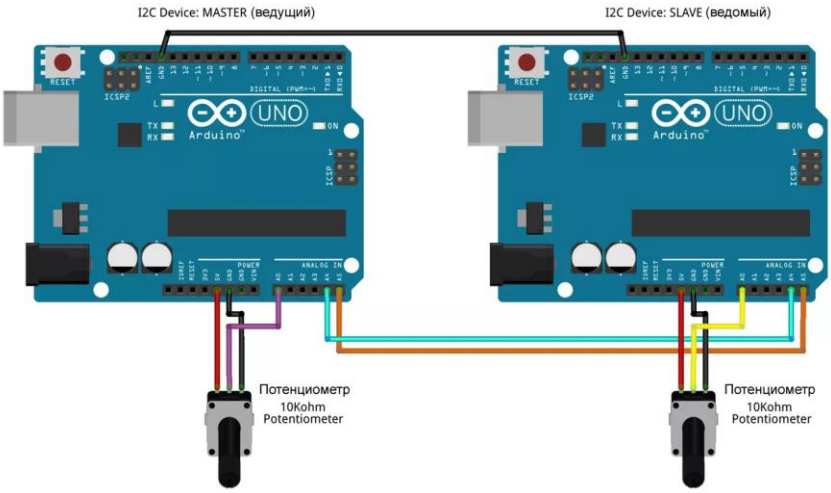

Рисунок 3 – Схема соединения двух Arduino с использованием I2C

Поскольку подтягивающие резисторы уже встроены в I2C контакты Arduino, то на линиях SDA и SCL нет необходимости во внешних подтягивающих резисторах.

Для использования встроенного интерфейса I2C Arduino рекомендуется использовать библиотеку Wire.

Эта библиотека поставляется в стандартной комплектации с Arduino IDE. Как и в других библиотеках Arduino, библиотека Wire имеет готовые I2C функции.

Опишем код для ведущего устройства:

```
// MASTER

#include <Wire.h>

byte i2c_rcv;                 // принятые данные
unsigned long time_start;     // start time in mSec
int stat_LED;                 // состояние светодиода 1 = ON, 0 = OFF
byte value_pot;               // положение потенциометра

void setup()
{
      Wire.begin(); // включить i2C в режим ведущего
```

```
      // Глобальные переменные
      i2c_rcv = 255;
      time_start = millis();
      stat_LED = 0;

      pinMode(13, OUTPUT);     // установить пин 13 на выход
}
```

Первое, что нужно сделать, это подключить устройство к шине I2C. Синтаксис для этого - Wire.begin(address). Адрес является необязательным для мастер-устройств. Итак, для эскиза мастера Arduino, мы просто добавляем код Wire.begin(); внутри setup().

Теперь перейдём к функции loop(). Сначала необходимо прочитать значение потенциометра, подключенного к контакту A0, и сохранить его в переменной value_pot.

```
void loop()
{

      value_pot = analogRead(A0);    // прочитать положение потенциометра

      // отправить слейву по адресу 0x08
      Wire.beginTransmission(0x08); // информируем шину о передаче данных на
ведомый с адресом 8 (0x08)
      Wire.write(value_pot);         // отправляем данные потенциометра
      Wire.endTransmission();        // информируем шину о прекращении передачи
      Wire.requestFrom(0x08, 1);     // получаем данные о потенциометре
ведомого
      if(Wire.available()) {         // Читаем ответ ведомого
            i2c_rcv = Wire.read();
      }

      // мерцаем всетодиодом
      if((millis() - time_start) > (1000 * (float)(i2c_rcv/255))) {
            stat_LED = !stat_LED;
            time_start = millis();
      }
      digitalWrite(13, stat_LED);

      }
```

После сохранения значения с пина A0 в переменную value_pot, мы можем отправить значение по I2C. Отправка данных по I2C включает в себя три функции:

- **Wire.beginTransmission();**
  Инициирует команду отправки, сначала информируя устройства на шине о том, что мы будем отправлять данные.

- **Wire.write();**
  Отправляет значение переменной value_to_send с помощью функции Wire.write(value)
- **Wire.endTransmission();**
  После отправки данных нам необходимо освободить сеть, чтобы позволить другим устройствам общаться по сети. Это делается с помощью функции Wire.endTransmission().

Полным синтаксисом запроса данных от ведомого устройства является Wire.requestFrom(адрес, количество байт).

Адрес - это I2C-адрес ведомого устройства, от которого мы должны получить данные, а количество - это количество байтов, которое нам нужно. Для нашего проекта, адрес ведомого устройства 0x08 и нам нужен один байт.

Внутри loop() мы используем Wire.requestFrom(0x08, 1); для запроса одного байта данных от ведомого устройства 0x08.

После выдачи команды Wire.requestFrom(0x08, 1), за ней должна следовать команда чтения для получения ответа от шины I2C.

Для получения данных от ведомого необходимо проверить, есть ли данные на шине. Это делается с помощью функции Write.available() внутри условного оператора if(). Функция Write.available() возвращает количество байт, ожидающих чтения.

Для получения доступных данных используем функцию Wire.read() с сохранением возвращаемого значения в переменную i2c_rcv. Каждый вызов функции Wire.read() получает только один байт данных из шины I2C.

Разберём код для ведомого устройства.

Для slave существует небольшая разница в кодировании I2C. Первая разница заключается в адресе Wire.begin(address).

Для ведомых устройств адрес является обязательным. Для нашего проекта адрес для ведомого устройства будет 0x08. Это может быть любой адрес, но убедитесь, что он уникален в сети I2C.

Некоторые I2C ведомые устройства также имеют определенные I2C-адреса, поэтому сначала проверьте спецификацию.

Мы присоединим устройство по I2C в качестве ведомого устройства, добавив код Wire.begin(0x08); внутри setup().

```
// SLAVE

#include <Wire.h>

byte i2c_rcv;                // data received from I2C bus
unsigned long time_start;    // start time in mSec
int stat_LED;                // status of LED: 1 = ON, 0 = OFF
byte value_pot;              // potentiometer position

void setup()
{
      Wire.begin(0x08);           // join I2C bus as Slave with address 0x08

      // event handler initializations
      Wire.onReceive(dataRcv);    // register an event handler for received
data
      Wire.onRequest(dataRqst);   // register an event handler for data
requests

      // initialize global variables
      i2c_rcv = 255;
      time_start = millis();
      stat_LED = 0;

      pinMode(13, OUTPUT);        // set pin 13 mode to output
}
```

Следующая задача - добавить в наш код обработчики событий для управления данными, полученными с других устройств в I2C сети.

Обработчики событий - это части кода, которые управляют событиями, с которыми наше устройство, скорее всего, столкнется во время работы.

В части скетча setup() мы добавляем функцию Wire.onReceive(handler) для регистрации функции (обработчика), которая будет управлять полученными данными.

Мы вызываем нашу функцию-обработчик dataRcv(). Обратите внимание, что имя функции может быть любым. В приведенном выше эскизе Wire.onReceive(dataRcv); вызывается в секции setup().

В конце эскиза находится код функции-обработчика. Он инициализируется как void dataRcv(int numBytes).

Параметр int numBytes содержит количество байт полученных данных.

```
void loop()
{

      value_pot = analogRead(A0); // read analog value at pin A0
(potentiometer voltage)

      // blink logic code
        if((millis() - time_start) > (1000 * (float)(i2c_rcv/255))) {
            stat_LED = !stat_LED;
```

```
            time_start = millis();
      }
      digitalWrite(13, stat_LED);
}

//received data handler function
void dataRcv(int numBytes)
{
      while(Wire.available()) {     // read all bytes received
            i2c_rcv = Wire.read();
      }
}

// requests data handler function
void dataRqst()
{
      Wire.write(value_pot); // send potentiometer position

      }
```

Следующий обработчик события, который используется - Wire.onRequest(handler). Эта функция крутится на подчиненных устройствах и работает аналогично Wire.onReceive().

Единственное отличие заключается в том, что она обрабатывает события запроса данных. Запросы данных поступают от основных устройств.

В функцию setup() мы добавляем код Wire.onRequest(dataRqst);. А в конце программы добавляем функцию void dataRqst(). Обратите внимание, что обработчики Wire.onRequest() не принимают никаких параметров. Функция dataRqst() содержит только Wire.write().

Нам не нужны Wire.beginTransmission() и Wire.endTransmission(), потому что библиотека Wire уже обрабатывает ответы от ведомых устройств.

Используя Arduino IDE, загрузите скетч мастера Arduino в одну из Ардуино. Затем загрузите скетч наследника в другую Arduino.

Отрегулируйте потенциометр на ведущем устройстве, чтобы регулировать частоту мигания светодиода ведомого устройства.

Отрегулируйте потенциометр на ведомом устройстве, чтобы контролировать частоту мигания светодиода ведущего устройства.

Наш код принимает положение потенциометра мастера и посылает его ведомому устройству через I2C. Затем ведомое устройство использует полученное значение для настройки времени задержки мигания светодиода. То же самое происходит и с положением потенциометра ведомого.

Практическая часть

1. Используя библиотеку Wire передать числовую информацию (двузначное целое число), введённую в консоли с ведущего на два ведомых устройства. Ведомые должны отобразить на семисегментном индикаторе разряд, соответствующий порядковому номеру ведомого: первый выводит младший разряд числа, второй выводит старший разряд.

2. Используя пример из теоретической части написать скетч, позволяющий по I2C управлять углом поворота сервопривода, подключенного к ведомому контроллеру, устанавливая угол поворота при помощи потенциометра на ведущем устройстве. (Для работы с сервоприводом использовать библиотеку Servo)

3. Собрать схему и написать программу двух авометров (измерителей тока и напряжения), опрашиваемых по шине I2C. Авометры должны опрашиваться Arduino в режиме мастера с выводом тока и напряжения в консоль.