Лабораторная работа № 7

Изучение 1-Wire

Цель лабораторной работы: познакомиться с принципами организации 1-Wire, а также научиться использовать 1-Wire на платформе Arduino.

Теоретическая часть

Интерфейс 1-Wire был разработан компанией Dallas Semiconductor в конце двадцатого века, однако позже компания Dallas Semiconductor была выкуплена корпорацией Maxim Inc. Широкого и повсеместного применения (как, например, I2C и SPI) он не получил, тем не менее интерфейс привлекает необходимостью использования всего лишь одного провода данных (и GND). Несмотря на то, что интерфейс асинхронный (не имеет тактового сигнала), временные рамки импульсов имеют гигантский гистерезис (на практически все временные рамки имеется лишь минимальное значение импульса) и не требуют точного соблюдения таймингов (как в случае с UART).

Сам интерфейс, как аппаратный, редко встречается в микроконтроллерах, но относительно легко реализуется программно. Подключение датчиков и прочих устройств происходит по схеме «монтажное ИЛИ», соответственно в данный момент времени успешно может проходить только одна транзакция данных и только в одном направлении.

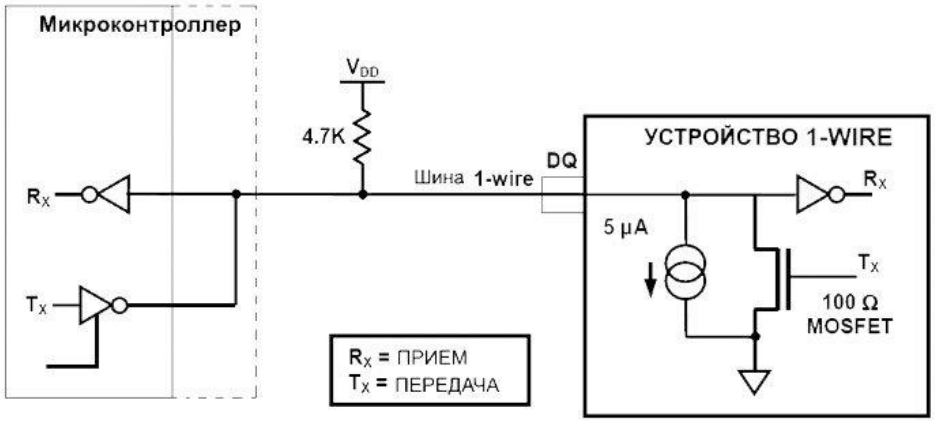

Рисунок 1 – Схема аппаратной части 1-Wire

На рисунке 1 показана упрощенная схема аппаратной составляющей интерфейса 1-Wire. Вывод данных OneWire-устройства представляет собой вывод структуры КМОП (Комплементарный Металл-Оксид-Полупроводник, также CMOS – Complementary Metal-Oxide-Semiconductor), который может быть притянут транзистором к земле питания. Но транзистор не будет полностью открыт, сопротивление его канала будет примерно равно 100 Ом. В замкнутом состоянии транзистора будет небольшой ток утечки, которым можно пренебречь.

С помощью транзистора шина может быть притянута **только к земле**. Верхняя часть моста отсутствует, то есть вывод представляет собой открытый коллектор. Для того, чтобы обеспечить стабильную логическую единицу на выводе, используется подтягивающий вверх резистор номиналом от 300 до 30000 Ом. Его номинал выбирается исходя из длины линии, а также исходя из максимального тока ножки устройства, которое собирается подтягивать вниз линию данных. Стандартным значением на небольшие линии данных является 4.7 кОм, напряжение питания – 3,3 – 5В (стандартные значения для большинства OneWire-устройств).

На рисунке 1 можно видеть два варианта подключения шины данных к микроконтроллеру – с использованием одного двунаправленного пина или же с использованием двух разнонаправленных линий ввода-вывода. У каждого варианта есть свои плюсы и свои минусы, например, с двумя линиями легче работать, но не всегда можно найти и выделить два пина, что особенно актуально для небольших корпусов с малым количеством ножек.

Принципы программной реализации интерфейса

1. Обмен всегда инициируется ведущим устройством, в нашем случае микроконтроллером.
2. Интерфейс предусматривает hot plug, то есть «горячее» подключение и отключение устройств от шины.
3. Обмен всегда начинается с импульса «reset», который генерируется мастером.
4. Обмен ведётся тайм-слотами (промежутками времени для передачи одного бита информации).
5. Данные передаются, начиная с менее значимого (младшего) бита.
6. Каждый пакет данных сопровождается контрольной суммой.

Обмен данными выглядит следующим образом:

1. Мастер шины посылает сигнал RESET.
2. Если на шине есть хотя бы одно устройство, оно отвечает на импульс, посылая свой импульс PRESENCE (этот же импульс отсылает устройство после подачи питания, получается power-upreset)
3. Ведётся передача данных
4. Передача RESET мастером (либо для досрочного прекращения обмена, либо для завершения сеанса)

Рисунок 2 - Временная диаграмма сигналов RESET и PRESENC

Импульс RESET генерируется мастером – мастер прижимает шину к земле и держит её не менее **480 мкс**, после чего отпускает шину, переводя вывод в высокоимпедансное состояние. Благодаря подтягивающему резистору, шина возвращается в логическую единицу, причём сделать это она должна не более, чем за **20 мкс**. Эта величина во многом определяет необходимое сопротивление подтягивающего резистора, ведь он, вместе с ёмкостью линии данных, создаёт своеобразную RC-цепочку, а ёмкость линии, как правило, поменять сложно.

По импульсу RESET ведомое устройство приводит себя в готовность и не позже **60 мкс** отвечает мастеру импульсом присутствия (PRESENCE), удерживая линию у земли на **60-240 мкс**. После выдачи импульса устройство отпускает шину.

Вне зависимости от длин импульсов RESET и PRESENCE, ведущее устройство должно начать обмен данными не ранее, чем через **480 мкс** после подачи импульса сброса. Такой промежуток времени даётся обеим сторонам для подготовки к обмену.

Таким образом, вся инициализация передачи вместе со всеми задержками и импульсами занимает **960мкс**.

Передача данных через тайм-слоты

Тайм-слот – отрезок времени, последовательность логических уровней сигнала, которая используется для передачи или приёма одного бита информации по шине OneWire.

Различают **4 вида тайм-слотов**:

- передача от ведущего;
- приём ведущим;
- передача от ведомого;
- приём ведомым.

Как и любой процесс на шине, тайм-слот инициируется ведущим устройством. Мастер шины переводит линию данных в низкий логический уровень и держит её в таком состоянии **от 60 до 120 мкс**. Между любыми двумя тайм-слотами должна быть пауза не менее **1 мкс**, подробнее необходимо уточнить у производителей микросхем в datasheets.

Важно понимать, что тайм-слоты передачи разительно отличаются от тайм-слотов приёма, так как в первом случае ведущий просто выдаёт данные на линию, а во втором случае работает в двунаправленном режиме (генерирует тайм-слот и читает данные от ведомого).

На рисунке 3 показаны диаграммы тайм-слотов для передачи и для приёма.

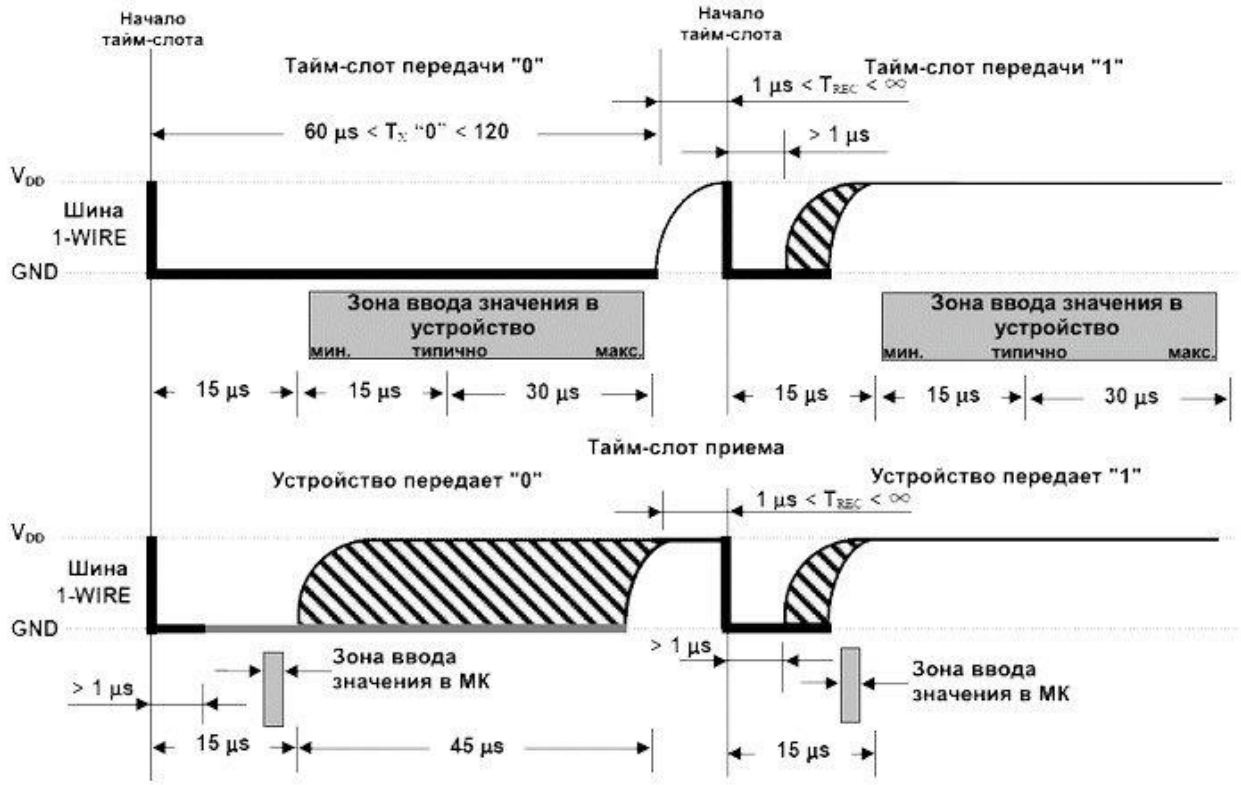

Рисунок 3 – Диаграммы тайм-слотов для передачи и приёма

Рассмотрим передачу данных через тайм-слот. Передача «0» заключается в удержании шины у земли в течение всего тайм-слота. Передача же единицы происходит так же с точки зрения инициации (60 – 120 мкс у земли), но устройство должно отпустить шину, выждав при этом не менее **1 мкс** и не более **15 мкс** с начала тайм-слота. Принимающее устройство считывает уровень, начиная с **30 мкс** и заканчивая **60 мкс** после начала тайм-слота (этот промежуток показан серым прямоугольником на рисунке 3). Для большинства устройств OneWire характерным временем считывания является **30 мкс**.

На рисунке 3 заштрихованной областью показана область гистерезиса, размер которой зависит от ёмкости шины и подтягивающего резистора.

Тайм-слот приёма ведущим отличается лишь тем, что после подачи инициирующего сигнала (**60мкс**) управление шиной на себя берёт передающее устройство (slave), а ведущий лишь анализирует шину на предмет битов в тайм-слотах. Все тайминги сохраняются – ведомое так же выставит уровень через **15 – 30 мкс**, а ведущий считает его. Гистерезис показан штрихами. Фактически, для уверенного приёма информации бит следует считать через **14 – 15 мкс** после его выставления. Этот параметр может изменяться, но рекомендуется опираться на наихудший из возможных вариантов для считывания любого возможного сигнала.

Ведущее устройство начинает обмен с ведомым с установления шины в «0» в течение **1 мкс**. Этот этап одинаков и для передачи, и для приёма. Далее, если ведущее устройство передаёт «1», то оно отпускает линию, если же «0» – держит её до конца тайм слота (**60 – 120 мкс**). Если мастер принимает данные, то на **13-15 мкросекунде** тайм-слота он должен опросить шину на предмет принимаемого бита. После каждого тайм-слота ведущий обеспечивает микросекундную задержку.

Необходимо учесть тот факт, что будучи асинхронной шиной, OneWire всё-таки очень важны тайминги. Если «передержать» линию у земли в тайм-слоте или в паузе, то ведомое может принять этот «0» за сигнал сброса линии и досрочно прекратить обмен. Но программная составляющая точных таймингов играет такую же роль, как и аппаратная. Поэтому разработка точных и гарантированно работающих устройств с применением OneWire получается чуть сложнее, чем разработка устройств на синхронных интерфейсах.

Транспортный уровень протокола

Любой протокол передачи данных содержит две части: низкоуровневая передача обеспечивается так называемым физическим протоколом, тогда как далее будет рассмотрена высокоуровневая передача, когда биты будут складываться в байты информации, а байты – в пакеты данных.

По стандарту, каждое устройство 1-Wire имеет свой уникальный **64-битный ID** (IDentification number), который будет являться уникальным «на протяжении 20 лет при сохранении текущих темпов производства», то есть, фактически, 100% уникальным. Этот номер вбивается в устройство ещё на этапе производства.

Для рассмотрения мы возьмём пример, в котором на шине имеется один мастер и n подчинённых устройств, причём $n > 1$. В таком случае, ведущее устройство должно сначала определить количество ведомых устройств, затем узнать их адреса на шине и только потом начинать обмен полезными данными.

Допустим, что нам известны все ID всех n устройств на шине, отсюда алгоритм работы будет следующий:

1. Импульс RESET
2. Ожидание ответного PRESENCE
3. Подача широковещательной команды
4. Индивидуальное получение данных от каждого из устройств
5. Импульс RESET

В спецификации 1-Wire оговорены команды, общие для всех сертифицированных OneWire-устройств (Таблица 1).

Таблица 1 – Основные команды 1-Wire устройств

Команда	**Значение**	**Описание команды**
SEARCH ROM	0xF0	Выполнение поиска адресов всех устройств на шине, используется в «сложном» алгоритме, когда количество и адреса устройств неизвестны.
READ ROM	0x33	Если на шине находится одно устройство, то этой командой можно узнать его адрес.
MATCH ROM	0x55	При наличии множества устройств на линии позволяет выбрать для обмена одно с конкретным адресом
SKIP ROM	0xCC	Ещё один способ адресации – если достоверно известно, что на шине находится одно устройство, то можно попросить его «игнорировать» адрес и отвечать на любой запрос.

После того, как ведущее устройство выдаст в шину команду READROM, ведомое отдаст свой собственный 8-байтный адрес, который должен принять ведущий. Любой обмен информацией должен быть прерван или завершён импульсом RESET. После выдачи команды MATCHROM, ведомые будут ждать от ведущего 8 байт адреса устройства, с которым планируется вести обмен информацией. Все устройства на шине, приняв команду MATCHROM, ждут появления байтов адреса и сравнивают его со своим. Таким образом, остаётся одно устройство, адрес которого совпал с требуемым, а остальные отпускают линию до появления следующего сигнала RESET. Далее мы можем посылать выбранному устройству всевозможные команды, предусмотренные его спецификацией, и другие устройства на шине не будут препятствовать обмену.

Если же на шине только одно устройство, то можно не усложнять процесс и послать команду SKIPROM. Если устройство получает эту команду, оно сразу же начинает считать, что выбрано именно оно, и будет воспринимать все команды на линии. Однако, бывают такие операции, когда эта команда может пригодиться не только для одного устройства, например, если на линии висит несколько термометров DS18B20, то сигнал начала

преобразования температуры можно послать всем устройствам сразу, а считывать потом, индивидуально, по адресам.

Напоминаем, что данные всегда передаются с младшего (менее значимого) бита, младшим байтом вперёд.

Структура ID 1-Wire устройства

Номер состоит из 64 бит информации, а именно:

- 1 байт кода семейства,
- 6 байт самого адреса
- 1 байт контрольной суммы CRC (Cyclic Redundancy Check).

Рассмотрим номер с самого младшего байта – контрольной суммы.

Контрольная сумма – специальный байт, который формируется из предыдущих байт таким образом, что если при передаче байт не было допущено ошибки, то CRC, отправленная в пакете данных, совпадёт с CRC, рассчитанной на принимающем устройстве. Необходимо обязательно учитывать этот факт и реализовать на микроконтроллере алгоритм, который рассчитает эту сумму и передаст её вместе с пакетом данных или примет от ведомого и сравнит с рассчитанной. Если CRC, принятая и посчитанная, не совпадают, то считается, что пакет доставлен с ошибкой и обмен прерывается.

После контрольной суммы идёт 6 байт адреса, который является уникальным и однозначно определяет устройство, с которым осуществляется взаимодействие.

Последний байт – код семейства. Этот байт содержит в себе информацию о типе и назначении устройства, подключенного к шине; таким образом, при подключении неопознанного устройства, можно узнать, что это за устройство и даже узнать некоторые его функциональные особенности (Таблица 2).

Таблица 2 - Часто использующиеся коды семейств

Код семейства (HEX)	**Устройства** ()iButton Package	**Оригинальное описание** (Memory size in bits unless specified)	**Описание**
01	(DS1990A)*, DS2401	1-Wire net address (serial number) only	Уникальный серийный номер-ключ
04	(DS1994), DS2404	4k NV RAM memory and clock, timer, alarms	4К энергонезависимого ОЗУ + часы, таймер и будильник
05	DS2405	Single addressable switch	Одиночный адресуемый ключ
06	(DS1993)	4k NV RAM memory	4К энергонезависимого ОЗУ
08	(DS1992)	1k NV RAM memory	1К энергонезависимого ОЗУ
09	(DS1982), DS2502	1k EPROM memory	1К электрически программируемого ПЗУ
0A	(DS1995)	16k NV RAM memory	16К энергонезависимого ОЗУ
0B	(DS1985), DS2505	16k EPROM memory	16К EEPROM
0F	(DS1986), DS2506	64k EPROM memory	64К EEPROM
10	(DS1920), DS1820, DS18S20	Temperature with alarm trips	Датчик температуры с сигнализатором отклонения
12	DS2406, DS2407	1k EPROM memory, two channel addressable switch	1К EEPROM + двухканальный адресуемый ключ.
14	(DS1971), DS2430A	256-bit EEPROM memory and 64-bit OTP register	256 бит EEPROM и 64 бита однократно программируемой памяти (ROM)
1A	(DS1963L)	4k NV RAM memory with write cycle counters	4К энергонезависимого ОЗУ со счетчиком числа циклов записи
1D	DS2423	4k NV RAM memory with external counters	4К энергонезависимого ОЗУ с внешним счетчиком
20	DS2450	4 channel A/D	Четырехканальный АЦП
24	(DS1904), DS2415	Real-time clock (RTC)	Часы реального времени
26	DS2438	Temperature, A/D	Датчик температуры, АЦП
2C	DS2890	Single channel digital potentiometer	Одноканальный цифровой потенциометр
30	DS2760	Temperature, current, A/D	Датчик температуры, тока, АЦП

Подключение датчика DS18B20 к микроконтроллеру

Самый распространенный вариант исполнения DS18B20 в корпусе TO-92.

Рисунок 4 – Внешний вид датчика температуры DS18b20

Выводы на схемах подключения указаны для такого варианта. Существуют две стандартные схемы подключения DS18B20 к микроконтроллеру:

Схема питания с внешним источником представлена на рисунке 5.

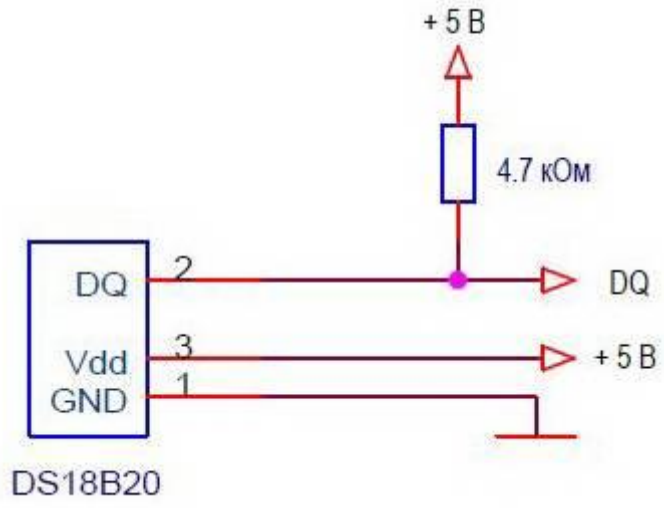

Рисунок 5 – Схема включения датчика DS18b20 с внешним источником питания

Подтягивающий резистор устанавливается вблизи вывода микроконтроллера.

Схема в режиме паразитного питания (рисунок 6) позволяет подключить термодатчик к микроконтроллеру всего двумя проводами, что особенно важно при размещении датчика на значительном расстоянии от контроллера.

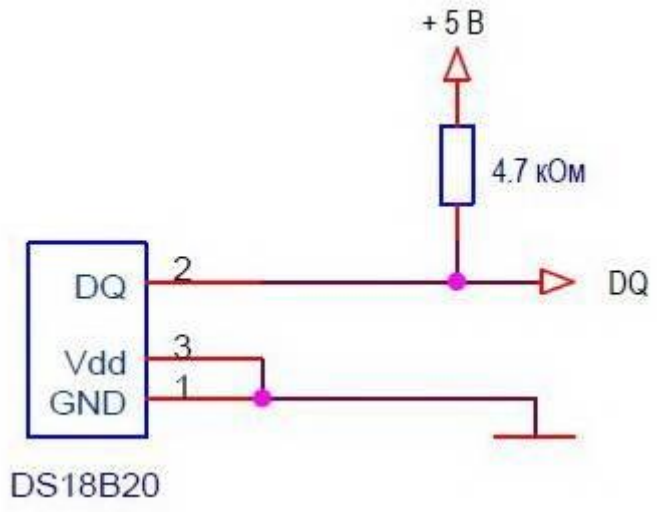

Рисунок 6 – Схема включения датчика DS18b20 в режиме паразитного питания

В этом режиме сигнал шины данных заряжает внутренний конденсатор датчика и за счет энергии на нем происходит питание устройства при низком уровне на шине. У режима паразитного питания существуют нюансы. Например, в этом режиме не гарантируется работа датчика при температуре свыше 100 °С. Надо использовать схему с внешним питанием.

Для такого случая я использую следующую схему (рисунок 7):

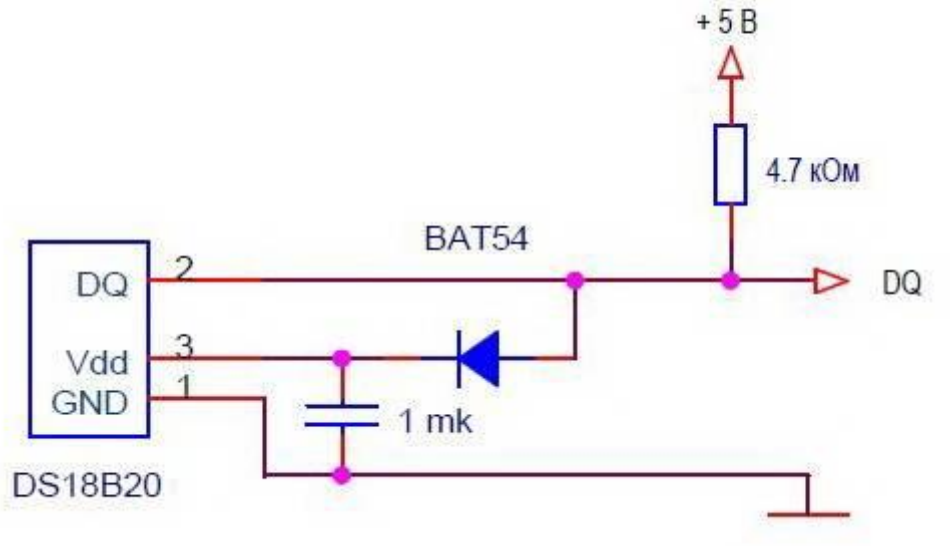

Рисунок 7 – Схема включения датчика с паразитным питанием и накоплением энергии

Датчик работает в режиме внешнего питания, которое запасается через диод на дополнительном конденсаторе.

Обмен информацией с термодатчиком

Необходимые операции работы с устройством:

- Инициализация – последовательность импульсов, с которых начинается любая операция на шине;
- Запись байта – передача байта данных в устройство DS18B20;
- Чтение байта – прием данных из устройства DS18B20.

Этих трех операций достаточно для работы с термодатчиком, все они поддерживаются библиотекой OneWire.

Библиотека Ардуино OneWire

OneWire(uint8_t pin), где

pin – номер вывода, к которому подключен датчик.

Например, формируем объект sensDs класса OneWire:

```
OneWire sensDs (14); // датчик подключен к выводу 14
```

uint8_t reset(void);

Инициализация операции на шине. С этой команды должна начинаться любая операция обмена данными. Возвращает:

- 1 – если устройство подключено к шине (был ответный импульс присутствия);
- 0 – если устройство отсутствует на шине (ответного импульса не было).

void write(uint8_t v, uint8_t power = 0);

Запись байта. Передает байт в устройство на шине.

- v – байт;
- power – признак выбора режима питания;
- power=0 – питание от внешнего источника
- power=1 – “паразитное” питание.

uint8_t read(void);

Чтение байта – принимает байт, переданный устройством. Возвращает значение принятого байта.

Этих команд достаточно, чтобы работать с датчиком DS18B20.

Можно добавить методы записи и чтения блоков байтов и функцию вычисления контрольной суммы.

void write_bytes(const uint8_t *buf, uint16_t count, bool power = 0);

Запись блока байтов.

- buf – указатель на массив;
- count – число байтов;
- power – признак выбора режима питания.

void read_bytes(uint8_t *buf, uint16_t count);

Чтение блока байтов.

- buf – указатель на массив;
- count – число байтов.

static uint8_t crc8(const uint8_t *addr, uint8_t len);

Функция вычисления 8ми разрядной контрольной суммы.

- addr - указатель на массив данных;
- len – число байтов.

Возвращает вычисленную сумму.

Последовательность операций работы с DS18B20

Для измерения температуры необходимо выполнить следующую последовательность действий:

- **sensDs.reset()** - Инициализация. Выполняет сброс шины, готовит ее для выполнения новой операции.
- **sensDs.write(0xCC, power)** - Команда пропуск ROM. У нас только один датчик на шине. Поэтому нет необходимости в поиске устройства с нужным адресом. Мы эту операцию пропускаем.
- **sensDs.write(0x44, power)** - Команда инициирует измерение температуры.
- **Пауза 1 сек**. Ожидание на время, необходимое для выполнения датчиком преобразования температуры. Это время зависит от выбранной разрешающей способности датчика. Мы используем максимальное разрешение 12 бит. Это разрешение установлено в датчике по умолчанию. Время преобразования для него – 750 мс. Если необходима другая разрешающая способность, то ее необходимо задать дополнительными командами.
- **sensDs. reset()** - Инициализация. Мы собираемся выполнить новую операцию на шине 1-Wire.
- **sensDs.write(0xCC, power)** - Команда пропуск ROM.

- **sensDs.write(0xBE, power)** - Команда чтения памяти датчика. Команда используется для чтения всех 9-ти байтов памяти DS18B20.

Таблица 3 – Байты памяти датчика DS18b20

Байт 0	Температура мл. байт
Байт 1	Температура ст. байт
Байт 2	Th регистр порога сигнализации
Байт 3	Tl регистр порога сигнализации
Байт 4	Регистр конфигурации
Байт 5	Зарезервирован
Байт 6	Зарезервирован
Байт 7	Зарезервирован
Байт 8	Циклический код CRC

- **read_bytes(buf, 9) -** Чтение 9ти байтов данных.
- **crc8(addr, 8) -** Вычисление контрольного кода данных.
- Сравнение контрольного кода с принятым.

После этой последовательности операций значение температуры содержится в первых двух байтах массива из принятых 9-ти байтов.

Отрицательная температура записывается в дополнительном коде (рисунок 8). Младший разряд имеет вес 0,0625 °C

ТЕМПЕРАТУРА	ЦИФРОВОЙ КОД (двоичный)	ЦИФРОВОЙ КОД (Hex)
+125°C	0000 0111 1101 0000	07D0h
+85°C	0000 0101 0101 0000	0550h
+25.0625°C	0000 0001 1001 0001	0191h
+10.125°C	0000 0000 1010 0010	00A2h
+0.5°C	0000 0000 0000 1000	0008h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1000	FFF8h
-10.125°C	1111 1111 0101 1110	FF5Eh
-25.0625°C	1111 1110 0110 1111	FE6Fh
-55°C	1111 1100 1001 0000	FC90h

Рисунок 8 – Цифровой код температуры

Пример работы с шиной и датчиком температуры DS18B20

```
// Новый экземпляр шины 1-Wire на пине P3
var oneWire = new OneWire(P3);

// Находим подключенные устройства на шине 1-Wire и отображаем массив
var devices = oneWire.search();
print(devices);

// Все дальнейшие дейстия осуществляем с первым найденным
// на шине устройством devices[0];

// Сброс, выбор устройства и передача команды на получение температуры
oneWire.reset();
oneWire.select(devices[0]);
oneWire.write(0x44);

// Сброс, выбор устройства и передача команды на считывание регистров
oneWire.reset();
oneWire.select(devices[0]);
oneWire.write(0xBE);

// Считываем 9-ть регистров  и отображаем его
var regs = oneWire.read(9);
print(regs);
```

OneWire(pin); - Возвращает объект, для работы с протоколом 1-Wire на пине pin.

OneWire.reset(); - Отправляет сигнал сброса на все устройства. При обнаружении хотя бы одного устройства возвращает true, в противном случае — false.

OneWire.search([command]); - Осуществляет поиск устройств и возвращает массив, где каждый элемент — 64-х битный адрес ROM. Для поиска используется команда command. Если параметр command не передан или равен 0x00 — используется команда 0xF0.

OneWire.select(rom); - Выбирает устройство для последующего обмена данными. Параметр rom — 64-х битный адрес ROM.

OneWire.write(data, [power]); - Передает по протоколу 1-Wire один или несколько (массив) байт. Если необходимо запитать устройства — передается параметр true в качестве значения параметра power.

OneWire.read([count]); - Возвращает полученные по протоколу 1-Wire данные. Если параметр count не передан, функция возвращает один байт, иначе — Uint8Array.

Практическая часть

1. С использованием библиотеки OneWire получить данные с датчика температуры ds18b20 и отобразить их на трёхразрядном семисегментном индикаторе (старший разряд – знак температуры, два младших разряда – числовое значение температуры).

2. Развить задачу 1 на случай нескольких датчиков. Добавить в проект кнопку, для выбора датчика: при каждом её нажатии на дисплей должна выводиться температура следующего датчика.

3. Разработать схему и написать программу термостата для аквариума, позволяющего задавать температурный гистерезис срабатывания реле. Данные о температуре и режиме работы необходимо вывести на LCD дисплей 1602. С использованием нескольких кнопок организовать возможность изменения верхнего и нижнего порогов температур.