**Протокол SmartHomeCAN.**

Коротко про шину CAN. Физика шины представляет собой витую пару проводников. Требования к кабелю схожи со стандартом RS485. Отлично работает компьютерная «витуха» UTP 5e. В CAN шине обмен данными происходит в виде коротких сообщений или фреймов (кадров, по англ. «frame»). Каждый фрейм имеет идентификатор (заголовок) - ID . Размер заголовка может быть 11 бит (стандартный фрейм) или 29 бит (расширенный). Тело фрейма (payload) это данные (до 8-ми байт информации). Служебные биты фрейма в данной статье не описываются. Шина мультимастерная, т.е. любой участник может начать передачу в любой момент времени. Передача широковещательная (все участники видят сообщения всех участников). Арбитраж выполняется при помощи CAN контроллера, посредством Rx и Tx буферов. Если шина занята в данный момент (определяется данное обстоятельство по факту проигрывания арбитража отправляемого сообщения), контроллер ждёт освобождения шины, храня отправляемое сообщение в буфере, и, как только шина становится свободна, возобновляет передачу. Если шина загружена и создаётся ситуация, когда одновременно выполняют передачу несколько участников, физика шины позволяет выигрывать арбитраж тем фреймам, у которых ID имеет наименьшее значение. Таким образом, коллизии практически исключаются аппаратным методом.

Сам по себе протокол CAN определяет всего лишь, как малые пакеты данных можно безопасно переместить из точки A в точку B посредством коммуникационной среды. Он, как и следовало ожидать, ничего не говорит о том, как управлять потоком; передавать большое количество данных, нежели помещается в 8–байтное сообщение; ни об адресах узлов; ни о назначении тех или иных идентификаторов фреймов; ни об установлении соединения и т.п. Эти пункты определяются протоколом более высокого уровня.

Таким и является наш предлагаемый протокол SmartHomeCAN. Он служит для организации сети контроллеров умного дома и создавался поверх стандартного протокола CAN шины. В протоколе будут использоваться только ID 29 бит. Ключевыми элементами созданного протокола являются – собственно, сами участники сети - микроконтроллеры (МК), подключенные к ним датчики и исполнительные механизмы. Чтобы упорядочить где что находится, в протоколе выполнена:

- адресация узлов CAN сети контроллеров;

- адресация «параметров» (параметры - это текущие данные с датчиков - метрики) ;

- адресация исполнительных механизмов (называем их тут «девайсами»).

Размер адресации узлов CAN сети имеет один байт. Поэтому сеть может иметь до 254 участников. «0» – это широковещательный адрес. Размер адресации «параметров» и «девайсов» также один байт. Поэтому и список параметров и девайсов каждый может быть в количестве от 1 до 254. Понятие «параметр» не является наименьшим объектом данных. Мы назвали это понятие «**тип** параметра», потому что у каждого типа параметра может быть несколько датчиков (это понятие мы назвали «№ датчика»).Например, *концевик открытия окна* – это «тип параметра», а *номер окна* – это «№ датчика». Скорее всего такой же подход будет и к исп. механизмам. Т.е. будет **«**тип девайса**»** и «№ девайса**»**. Таким образом, адресация доступа к параметру (девайсу) будет тройная: *адрес узла -> адрес типа параметра(девайса) -> № датчика (девайса).* В случаях когда «тип параметра» или «девайса» не делится на датчики, а сам является наименьшим объектом, № датчика будет равен «1» .

Все передаваемые по сети фреймы (сообщения) имеют свой уникальный номер (называем его «глобальный счётчик») и подразделяются на следующие типы:

*- Команда*

*- Отчет о получении команды и о результате её выполнения*

*- Запрос параметра*

*- Предоставление параметра (по запросу/ по его изменению/периодически)*

*- Запрос статуса узла*

*- Ответ на запрос статуса*

*- Запрос мультикадра*

*- Мультикадр*

*- Конец мультикадра*

*- Авария*

*- Отчет о принятии аварии*

Разберём каждый тип сообщения более подробно (более наглядная информация будет предоставлена в картах типов сообщений в приложении).

**Команда** (COMMAND) - управляет исполнительными механизмами (девайсами) и не только. Команду может посылать любой узел любому узлу. Алгоритм протокола здесь действует следующим образом. После посылки команды, отославший её МК ждёт отчета от целевого узла, если отчёт не приходит в течение определённого времени (по умолчанию 200мс), данный МК шлёт команду повторно, причём с тем же глобальным счётчиком. Тем самым исключается факт двойного выполнения команды (в том случае, если целевой узел при первой посылке всё же команду получил), т.к. целевой узел НЕ реагирует на команды, которые прилетают с тем же глобальным счётчиком в течение определённого времени. Такой алгоритм повторения команды (при неполучении отчёта) будет происходить до 5 раз, после чего, в случае всё равно неудачи, сохраняется код неисправности и жалоба в отладку.

Команды тоже подразделяются на типы. Большинство типов команд – это управление исполнительными механизмами (девайсами) - происходит путем изменения значений GPIO, к которым подключен соответствующий девайс. (digital, PWM). Но это не единственное, что могут команды. Также можно при помощи определённого типа команды *PARAMETER\_WRITE* изменить значение любого из списка параметров массива *parameter.* Поэтому у каждого типа команды выполнена возможность иметь до 4 байт значений. Такой подход позволяет передавать числовые значения размером до 4 байт. (32 битные числа). Удаленное изменение значений массива «параметров» делает не жесткую привязку подключения девайсов к GPIO. Например, возьмём такой девайс, как модуль света, который работает по своей шине DMX512 (часто по этой шине работает концертное, театральное оборудование). В общем и целом, нас даже не интересует как при этом подключен девайс к узлу, главное, что его работа организована от определённой переменной. Пусть в нашем случае это будет значение яркости лампы с количеством градаций в один байт (0…255). Яркость лампы данный девайс будет «брать» из соответствующего сконфигурированного нами на узле «типа параметра» (ну и далее уже по-своему отправляет в шину DMX512 на модуль самой лампы значение яркости, а как, нас, собственно, мало интересует). Таким образом, изменяя удаленно (с другого узла) значение переменной этого «типа параметра», мы можем извне управлять этим девайсом (модулем света DMX512) не через изменение значений GPIO, а универсально, изменив значение переменной, с которой работает девайс. Так вот просто, вы адаптируете свой девайс к CAN сети, а уж по взаимодействию девайса с узлом (аппаратное подключение и программный код) наш протокол SmartHomeCAN в данном случае никаких ограничений не накладывает. Тут работа за вами. Такой подход задаёт универсальность и возможность адаптировать к CAN сети любой девайс или датчик.

Типы команд на данный момент:

*RESERVE, 00 - Резерв*

*DIGITAL\_REMOTE 01 - Включение/выключение булевых устройств*

*DIGITAL\_INVERT, 02 - Инвертирование состояния булевых устройств*

*DIMMER\_SETTING, 03 - Установка процента включения диммируемых устройств (0…100%)*

*DIMMER\_TURN\_OFF, 04 - Уменьшение процента диммера (0…100%)*

*DIMMER\_TURN\_ON, 05 - Увеличение процента диммера (0…100%)*

*IMPULSE\_REMOTE, 06 - Включение/выключение импульсных устройств*

*IMPULSE\_INVERT, 07 - Инвертирование состояния устройств, , управление которыми 1 сек импульсом GND*

*PWM\_SETTING, 08 - Установка уровня диммируемых устройств PWM (0…255)*

*PWM\_TURN\_OFF, 09 - Уменьшение уровня PWM устройств (0…255)*

*PWM\_TURN\_ON, 0A - Увеличение уровня PWM устройств (0…255)*

*PARAMETER\_WRITE, 0B - Изменение значения выбранного параметра (массив parameter)*

В будущем можно добавить такие типы команд, как запрос на предоставление кодов неисправностей на узле и другие (предлагайте).

**Отчет о полученной команде** (COMMAND\_REPORT) – в отчёте содержится информация на какую команду сформирован этот отчёт, и результат выполнения команды. Отчёт посылается тому узлу, который эту команду отправил, причём в отчете также указывается уникальный номер (глобальный счётчик) команды. Т.е. всегда будет понятно, на какую именно команду был сформирован отчет.

Виды отчетов о результате выполнения команды :

*COMPLETE 01 // команда выполнена успешно*

*EXECUTING 02 // команда принята, началось выполнение*

*FAIL\_UNKNOWN 03 // ошибка, причины неизвестны*

*UNKNOWN\_COMMAND 04 // ошибка, неизвестный тип команды*

*UNKNOWN\_TYPEDEV 05 // ошибка, неизвестный тип устройства (параметра), посланный в команде.*

*NOT\_INCLUDED 06 // ошибка, тип устройства (параметра), посланный в команде, отсутствует на узле*

*NOT\_SUITABLE 07 // ошибка, команда не подходит для данного вида подключения устройства.*

*NEGATIVE\_RESULT 08 // ошибка, желаемый результат воздействия на устройство не достигнут (желаемое состояние устройства после выполнения команды не соответствует значению , полученному при контроле состояния устройства)*

**Запрос параметра**. (PARAMETER\_REQUEST). Этот тип сообщения позволяет сделать запрос любого «типа параметра» любым узлом у любого узла. В запросе также содержится «№ датчика», который запрашиваем. За один запрос можно запросить до 7 датчиков. Также можно запросить датчик с номером «00» , тогда узел ответит сразу всеми датчиками, которые присутствуют у запрашиваемого «типа параметра».

**Параметр**. (PARAMETER). Это сообщение-метрика, один фрейм – один параметр (или датчик, если параметр делится на датчики). Также в теле данного сообщения содержится информация о причине отправки метрики:

*PERIODICALLY 09 // параметр отправлен периодически*

*CHANGEVALUE 0A // параметр отправлен по факту изменения его значения на узле*

*ON\_REQUEST 0B // параметр отправлен по запросу от другого узла*

*NOT\_INCLUDED 06 // запрашиваемый параметр или датчик не укомплектован на узле*

По функциям участники сети делятся на «узлы» , «мастера» и «слейва». В данном контексте «мастер» служит для контроля работоспособности узлов в сети (выполняя heartbeat) и для связи CAN сети с внешним миром (с другими сетями , MQTT , GSM, Ethernet и т.д.), т.е. является шлюзом (gateway). При этом, с точки зрения арбитража в сети, вся прелесть CAN шины сохраняется - все узлы являются мастерами, а именно, любой МК может начать передачу в любой момент времени любому узлу (или всем сразу).

У мастера в памяти храниться таблица комплектации СAN сети (т.е. какие узлы должны присутствовать в сети). Мастер раз в секунду широковещательным сообщением (broadcast) рассылает узлам текущее время. Это сообщение названо «запрос статуса» (STATUS\_REQUEST). Получив данное сообщение, узлы отвечают «статусом» (STATUS). При этом каждый узел отвечает с разной задержкой относительно момента получения STATUS\_REQUEST (величина которой зависит от его адреса), чтобы разнести ответы статусов по времени, не нагружая CAN шину. В случае «отваливания» от сети какого-либо узла мастер сохраняет код неисправности(пока не реализовано) и жалуется в отладку.

Понятие «Слейв» - задел на будущее (пока не реализовано). Функцией Слейва будет следить за работой мастера, и если тот потерял свою работоспособность , слейв должен перенять обязанности мастера на себя.

«Узел» - это, так сказать, рабочий участник CAN сети умного дома. Он как раз и делает полезную работу – опрашивает датчики, «шевелит» исполнительными механизмами и т.д.

Начав построение CAN-сети контроллеров умного дома, сначала нужно приступить к конфигурации общего файла для всех участников сети (Файл *name\_addresses.h)* В этом файле заполняются как раз три ключевых списка, применяемых конкретно в вашем умном доме:

- названия/адреса всех контроллеров (узлов) сети;

- названия/адреса всех возможных «типов параметров»;

- названия/адреса всех возможных «типов девайсов».

А также в этом же файле для мастера заполняется список - какими узлами укомплектована CAN сеть.

Т.к. этот файл будет общий для всех узлов сети, её участники будут в курсе и понимать эту адресацию.

Далее необходимо выполнить конфигурацию отдельно каждого контроллера:

1. В начале скетча при помощи #define выбирается адрес (из списка *name\_addresses.h*) данного контроллера, модель МК, функциональное назначение узла («мастер» или «узел»), управление отладкой в терминал;
2. В файле *param\_conf.h* заполняется массив *parameter[][].* В этом массиве заполняется информация:

- какие «типы параметров» из файла *name\_addresses.h* будут присутствовать на данном конкретном узле;

- какой тип данных имеет каждый «тип параметра» (uint\_8, 16 и т.д.);

- сколько датчиков имеет каждый «тип параметра»;

- на какие узлы должна происходить автоматическая периодическая отправка каждого «типа параметра» и «№ датчика».

1. В файле *devices\_conf.h* заполняется массив *device [][].* В этом массиве заполняется информация:

- какие «типы девайсов» из файла *name\_addresses.h* будут присутствовать на данном конкретном узле;

- номер пина МК, по которому осуществляется управление девайсом;

- вид управления девайсом (ШИМ, DIGITAL или импульсное);

- номер пина МК, по которому осуществляется контроль состояния девайса

- информация о том, долгое ли будет выполнение команды на девайсе или мгновенное.

**Приложение. Карты типов сообщений**

**Команда (COMMAND)**

Пример CAN фрейма команды:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | ID 29 бит | | | | | | 1бит  (0...1) | 4бита  (0…F) | 8бит  (0…FF) | 8бит  (0…FF) | 8бит  (0…FF) | | **Приоритет CAN фрейма** | **Тип сообщения** | Адрес отправителя | Адрес получателя | **Тип девайса (параметра)** | | **1** | **1** | 04 | 0А | **02** | | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Поле данных (payload) 8 байт | | | | | | | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | Глобальный счётчик | **Номер датчика** | Резерв | **Тип команды** | **Значение команды**  **байт 1** | **Значение команды байт 2** | **Значение команды байт 3** | **Значение команды байт 4** | | 34 | **01** | 00 | **01** | **01** | **00** | **00** | **00** | |

**Виды уровня приоритета CAN сообщения:**

0 – Высокий приоритет

1 – Низкий приоритет

**Типы сообщений:**

NULL\_C, 0 - Резерв

COMMAND, 1 - Команда

COMMAND\_REPORT, 2 - Отчет о получении команды и о результате её выполнения

PARAMETER\_REQUEST, 3 - Запрос параметра

PARAMETER, 4 - Предоставление параметра (по запросу/ по его изменению/периодически)

STATUS\_REQUEST, 5 - Запрос статуса узла

STATUS, 6 - Ответ на запрос статуса

MULTIFRAME\_REQUEST, 7 - Запрос мультикадра

MULTIFRAME, 8 - Мультикадр

MULTIFRAME\_END, 9 - Конец мультикадра

ACCIDENT, А - Авария

ACCIDENT\_REPORT, В - Отчет об аварии

**Типы команд.**

RESERVE, 00 - Резерв

DIGITAL\_REMOTE 01 - Включение/выключение булевых устройств

DIGITAL\_INVERT, 02 - Инвертирование состояния булевых устройств

DIMMER\_SETTING, 03 - Установка процента включения диммируемых устройств (значение в CommandValue)

DIMMER\_TURN\_OFF, 04 - Уменьшение процента диммера (значение увеличения в CommandValue)

DIMMER\_TURN\_ON, 05 - Увеличение процента диммера (значение уменьшения в CommandValue)

IMPULSE\_REMOTE, 06 - Включение/выключение импульсных устройств

IMPULSE\_INVERT, 07 - Инвертирование состояния импульсных устройств

PWM\_SETTING, 08 - Установка уровня диммируемых устройств PWM (значение в CommandValue)

PWM\_TURN\_OFF, 09 - Уменьшение уровня PWM устройств (значение увеличения в CommandValue)

PWM\_TURN\_ON, 0A - Увеличение уровня PWM устройств (значение уменьшения в CommandValue)

PARAMETER\_WRITE, 0B - Изменение значения выбранного параметра (массив parameter)

Для инициирования отправки узлом команды кому-либо нужно вызвать такую функцию

(параметры функции все однобайтовые):

*SendCommand (приоритет, тип команды, значение, значение\_2, значение\_3, значение \_4, адрес получателя, тип девайса, № датчика )*

**Отчёт с результатом выполнения команды (COMMAND\_REPORT)**

Пример CAN фрейма отчёта на принятую команду:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | ID 29 бит | | | | | | 1бит  (0...1) | 4бита  (0…F) | 8бит  (0…FF) | 8бит  (0…FF) | 8бит  (0…FF) | | **Приоритет CAN фрейма** | **Тип сообщения** | Адрес отправителя | Адрес получателя | **Тип девайса (параметра)** | | **1** | **2** | 04 | 0А | **02** | | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Поле данных (payload) 8 байт | | | | | | | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | Глобальный счётчик | **Номер датчика** | **Результат выполнения**  **команды** | **Тип команды** | Значение команды | Значение команды 2 байт | Значение команды 3 байт | Значение команды 4 байт | | 34 | **01** | **01** | **01** | 01 | 00 | 00 | 00 | |

**Виды уровня приоритета CAN сообщения:**

0 – Высокий приоритет

1 – Низкий приоритет

**Типы сообщений:**

NULL\_C, 0 - Резерв

COMMAND, 1 - Команда

COMMAND\_REPORT, 2 - Отчет о получении команды и о результате её выполнения

PARAMETER\_REQUEST, 3 - Запрос параметра

PARAMETER, 4 - Предоставление параметра (по запросу/ по его изменению/периодически)

STATUS\_REQUEST, 5 - Запрос статуса узла

STATUS, 6 - Ответ на запрос статуса

MULTIFRAME\_REQUEST, 7 - Запрос мультикадра

MULTIFRAME, 8 - Мультикадр

MULTIFRAME\_END, 9 - Конец мультикадра

ACCIDENT, А - Авария

ACCIDENT\_REPORT, В - Отчет об аварии

**Результат выполнения команды:**

COMPLETE 01 // команда выполнена успешно

EXECUTING 02 // команда принята, началось выполнение

FAIL\_UNKNOWN 03 // ошибка, причины неизвестны

UNKNOWN\_COMMAND 04 // ошибка, неизвестная команда

UNKNOWN\_TYPEDEV 05 // ошибка, неизвестный тип устройства, посланный в команде.

NOT\_INCLUDED 06 // ошибка, тип устройства, посланный в команде, отсутствует на узле

NOT\_SUITABLE 07 // ошибка, команда не подходит для данного вида устройства.

NEGATIVE\_RESULT 08 // ошибка, желаемый результат воздействия на устройство не достигнут (желаемое состояние устройства после выполнения команды не соответствует значению , полученному при контроле состояния устройства)

**Типы команд.**

RESERVE, 00 - Резерв

DIGITAL\_REMOTE 01 - Включение/выключение булевых устройств

DIGITAL\_INVERT, 02 - Инвертирование состояния булевых устройств

DIMMER\_SETTING, 03 - Установка процента включения диммируемых устройств (значение в CommandValue)

DIMMER\_TURN\_OFF, 04 - Уменьшение процента диммера (значение увеличения в CommandValue)

DIMMER\_TURN\_ON, 05 - Увеличение процента диммера (значение уменьшения в CommandValue)

IMPULSE\_REMOTE, 06 - Включение/выключение импульсных устройств

IMPULSE\_INVERT, 07 - Инвертирование состояния импульсных устройств

PWM\_SETTING, 08 - Установка уровня диммируемых устройств PWM (значение в CommandValue)

PWM\_TURN\_OFF, 09 - Уменьшение уровня PWM устройств (значение увеличения в CommandValue)

PWM\_TURN\_ON, 0A - Увеличение уровня PWM устройств (значение уменьшения в CommandValue)

PARAMETER\_WRITE, 0B - Изменение значения выбранного параметра (массив parameter)

В отчёте байт глобального счётчика выполняется равным глобальному счётчику принятой команды. Поэтому отправивший команду узел понимает, на какую конкретно команду формируется данный отчёт.

**Запрос параметра (PARAMETER\_REQUEST)**

Пример CAN фрейма запрос параметра:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | ID 29 бит | | | | | | 1бит  (0...1) | 4бита  (0…F) | 8бит  (0…FF) | 8бит  (0…FF) | 8бит  (0…FF) | | **Приоритет**  **CAN фрейма** | **Тип сообщения** | Адрес отправит. | Адрес получат. | **Тип девайса**  **(параметра)** | | **1** | **3** | 03 | 09 | **07** | | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Поле данных (payload) | | | | | | | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | Глобальный счётчик | **Номер датчика** | **Номер датчика** | **Номер датчика** | **Номер датчика** | **Номер датчика** | **Номер датчика** | **Номер датчика** | | 11 | **01** | **02** | **03** | **04** | **05** | **06** | **07** | |

**Виды уровня приоритета CAN сообщения:**

0 – Высокий приоритет

1 – Низкий приоритет

**Типы сообщений:**

NULL\_C, 0 - Резерв

COMMAND, 1 - Команда

COMMAND\_REPORT, 2 - Отчет о получении команды и о результате её выполнения

PARAMETER\_REQUEST, 3 - Запрос параметра

PARAMETER, 4 - Предоставление параметра (по запросу/ по его изменению/периодически)

STATUS\_REQUEST, 5 - Запрос статуса узла

STATUS, 6 - Ответ на запрос статуса

MULTIFRAME\_REQUEST, 7 - Запрос мультикадра

MULTIFRAME, 8 - Мультикадр

MULTIFRAME\_END, 9 - Конец мультикадра

ACCIDENT, А - Авария

ACCIDENT\_REPORT, В - Отчет об аварии

Как и во всех других типах сообщений нулевой байт поля данных это глобальный счётчик. Запрашиваемый тип параметра содержится в ID . Размер поля данных (DLC) может быть разный – в зависимости от того, сколько датчиков данного типа параметра хотим запросить. За один фрейм можно запросить до 7 датчиков.

- Если хотим запросить только один датчик (например когда тип параметра не делится на датчики и при этом имеет только датчик с номером 1), то размер поля данных (DLC) будет равен двум. Т.е. поле данных будет представлять собой два байта: глобальный счётчик, № датчика (1).

- Если хотим запросить сразу все имеющиеся датчики у данного типа параметра, то запрашиваем только один датчик с номером «0».

Чтобы инициировать отправку запроса параметра узлом, нужно вызвать в его скетче такую функцию (если хотим номера датчиков перечислить прямо в вызове функции, делаем так):

*SendRequestParam (приоритет, адрес получателя, тип параметра , Sensors(4,5,6));*

Фрейм запроса будет выглядеть так:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | | | | | PAYLOAD | | | |
| Приорит. | Тип сообщ. | Адрес отпр. | Адрес получ. | Тип параметра | Глоб. счётч. | Запраш. датч. | Запраш. датч. | Запраш. датч. |
| 1 | 3 | 03 | 09 | 07 | 12 | 04 | 05 | 06 |

Тут мы запросили датчики 4,5,6 типа параметра «07» у узла «09».

Если хотим номера датчиков запросить, закинув в вызов функции массив с номерами датчиков, делаем так):

*byte sensors[] = {1,4,8,9};*

*SendRequestParam (приоритет, адрес получателя, тип параметра , sizeof(sensors), sensors);*

Тут мы запросили датчики 1,4,8,9

Если хотим запросить сразу все датчики необходимого типа параметра, делаем так:

*SendRequestParam (приоритет, адрес получателя, тип параметра , Sensors(0));*

Фрейм при этом будет такой :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | | | | | PAYLOAD | |
| Приорит. | Тип сообщ. | Адрес отпр. | Адрес получ. | Тип параметра | Глоб. счётч. | Запраш. датч. |
| 1 | 3 | 03 | 09 | 07 | 13 | 00 |

**Предоставление параметра (PARAMETER)**

Пример CAN фрейма предоставление параметра:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | ID 29 бит | | | | | | 1бит  (0...1) | 4бита  (0…F) | 8бит  (0…FF) | 8бит  (0…FF) | 8бит  (0…FF) | | **Приоритет**  **CAN фрейма** | **Тип сообщения** | Адрес отправителя | Адрес получателя | **Тип девайса**  **(параметра)** | | **1** | **4** | 05 | 06 | **0A** | | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Поле данных (payload) | | | | | | | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | Глобальный счётчик | **Номер датчика** | **Причина отправки параметра** | **Тип данных** | **Значение 1 байт** | **Значение 2 байт** | **Значение 3 байт** | **Значение 4 байт** | | 24 | **01** | **02** | **03** | **04** | **05** | **06** | **07** | |

**Виды уровня приоритета CAN сообщения:**

0 – Высокий приоритет

1 – Низкий приоритет

**Типы сообщений:**

NULL\_C, 0 - Резерв

COMMAND, 1 - Команда

COMMAND\_REPORT, 2 - Отчет о получении команды и о результате её выполнения

PARAMETER\_REQUEST, 3 - Запрос параметра

PARAMETER, 4 - Предоставление параметра (по запросу/ по его изменению/периодически)

STATUS\_REQUEST, 5 - Запрос статуса узла

STATUS, 6 - Ответ на запрос статуса

MULTIFRAME\_REQUEST, 7 - Запрос мультикадра

MULTIFRAME, 8 - Мультикадр

MULTIFRAME\_END, 9 - Конец мультикадра

ACCIDENT, А - Авария

ACCIDENT\_REPORT, В - Отчет об аварии

**Причина отправки параметра:**

PERIODICALLY 09 // параметр отправлен периодически

CHANGEVALUE 0A // параметр отправлен по факту изменения его значения на узле

ON\_REQUEST 0B // параметр отправлен по запросу от другого узла

NOT\_INCLUDED 06 // запрашиваемый параметр или датчик не укомплектован на узле

**Типы данных:**

BYTE\_1 01 // беззнаковый 1-байтный (0...255)

BYTE\_2 02 // беззнаковый 2-байтный (0...65535)

BYTE\_4 04 // беззнаковый 4-байтный (0...4294967295)

BYTE\_1\_SIGNED 06 // знаковый 1-байтный (-128...+127)

BYTE\_2\_SIGNED 07 // знаковый 2-байтный (-32768...+23767)

BYTE\_4\_SIGNED 08 // знаковый 4-байтный (-2147483648...2147483647)

BYTE\_4\_FLOAT 09 // 4-байтный флоат, с плавающей точкой

Один датчик – один фрейм. Как и во всех других типах сообщений нулевой байт поля данных это глобальный счётчик. Предоставляемый тип параметра содержится в ID . Размер поля данных (DLC) может быть разный – в зависимости от того, какой тип данных имеет данный предоставляемый датчик данного типа параметра.

Например, фрейм предоставляемого по запросу типа параметра «0А» датчика «01» от узла «05» узлу «06» с типом данных «uint\_16» и значением «16862» будет выглядеть так:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | | | | | PAYLOAD | | | | | |
| Приоритет | Тип  сообщения | Адрес отправителя | Адрес  получателя | Тип  параметра | Глоб. cчётч | № датчика | Причина отправки | Тип данных | Значение  Байт 1 | Значение  Байт 2 |
| 1 | 4 | 05 | 06 | 0A | 25 | 01 | 0B | 02 | 41 | DE |