

SAE J1939 ECU Programming & Vehicle Bus Simulation with the Arduino

```
// Establish the timer base in units of milliseconds
delay(SYSTEM_TIME);

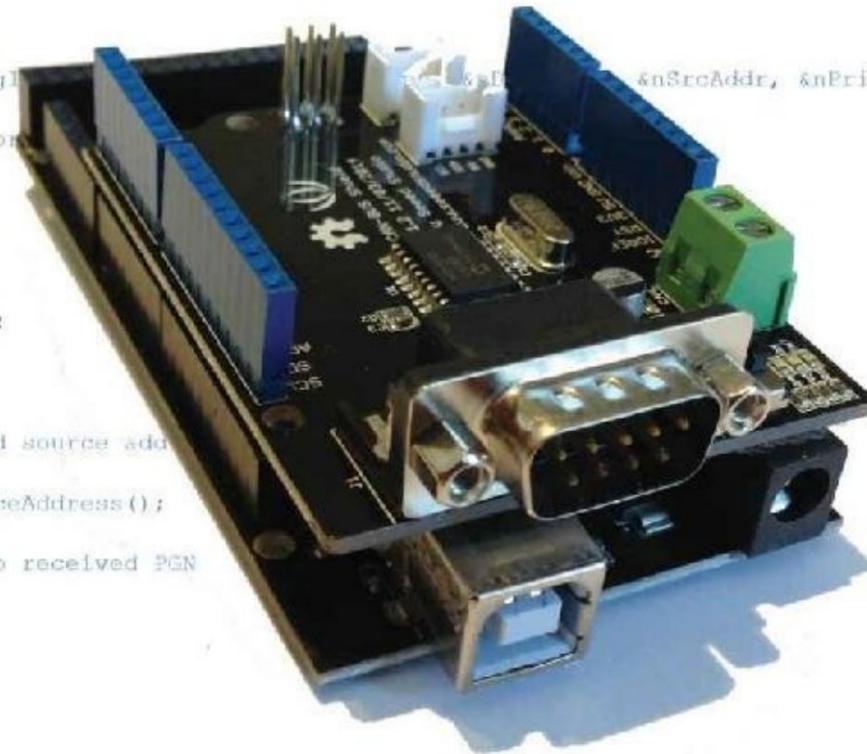
// Call the J1939 protocol stack
nJ1939Status = j1939Operate(&nMsg1,
                             &nMsg2,
                             &nMsg3,
                             &nSrcAddr, &nPriority);

// Check for reception of PGNs for
if(nMsgId == J1939_MSG_APP)
{
    // Check J1939 protocol status
    switch(nJ1939Status)
    {
        case ADDRESSCLAIM_INPROGRESS:
            break;

        case NORMALDATATRAFFIC:
            // Determine the negotiated source address
            BYTE nAppAddress;
            nAppAddress = j1939GetSourceAddress();

            // Respond corresponding to received PGN
            switch(IPGN)
            {
                // end switch(IPGN)
                break;

        case ADDRESSCLAIM_FAILED:
            break;
    }
}
```



Includes ARD1939, the SAE J1939 Protocol Stack
for Arduino Uno & Arduino Mega 2560

Wilfried Voss

Программирование ЭБУ SAE J1939 и автомобильная шина Симуляция с АРДУИНО

Уилфрид Восс

Авторское право © 2015 г. корпорация Copperhill Technologies <http://copperhilltech.com>

Все товарные знаки или авторские права, упомянутые здесь, являются собственностью их соответствующих владельцев, и Copperhill Media не претендует на право собственности в связи с упоминанием продуктов, содержащих эти знаки.

«Arduino» — торговая марка команды Arduino.

Примечание олицетворяет для чтения электронных книг

Эта электронная книга предназначена только для вашего личного пользования. Эта электронная книга не может быть продана или передана другим людям. Если вы хотите поделиться этой книгой с другим человеком, приобретите дополнительный экземпляр для каждого человека, с которым вы делитесь ею. Если вы читаете эту книгу и не купили ее, или она была куплена не только для вашего пользования, вам следует вернуть ее и приобрести собственный экземпляр. Спасибо за уважение к творчеству автора.

Ограничение ответственности/Отказ от гарантии

Хотя издатель и автор приложили все усилия для подготовки этой книги, они не делают никаких заявлений или гарантий в отношении точности или полноты содержания этой книги, в частности, отказываются от любых подразумеваемых гарантий, товарного состояния или пригодности для определенной цели. Никакие гарантии не могут создаваться или продлеваться торговыми предпринимателями или письменными торговыми материалами.

Советы и стратегии, содержащиеся здесь, могут не подходить для вашей ситуации. При необходимости следуйте инструкциям в соответствующем томе. Ни издатель, ни автор не несут ответственности за какие-либо убытки или прибыль или любые другие коммерческие убытки, включая, помимо прочего, обычные, случайные, косвенные или другие убытки.

ISBN-10: 1938581199

ISBN-13: 978-1-938581-19-9



<http://copperhilltech.com>

От автора

Прежде всего, приношу свои извинения всем, кто читает эту книгу с помощью электронных устройств для чтения (например, Kindle). Включение исходного кода для электронных ридеров невероятно сложно, и чтение на экранах определенных форматов может быть обременительным.

После написания контроллерной сети (CAN) с помощью Arduino (см. приложение к литературе в этой книге) с ледующим шагом было изучение протоколов более высокого уровня на основе CAN, в данном случае с использованием протокола транспортного средства SAE J1939. Я лично считаю J1939 наиболее эффективным протоколом CAN на рынке из-за его небольшого объема памяти в сочетании с пропускной способностью, которая легко преодолевает другие протоколы, такие как CANopen и DeviceNet.

Однако мой быстрый выбор SAE J1939 был основан не только на технических аспектах, но и на большой популярности протокола J1939 на рынке Северной Америки. CANopen и DeviceNet, принимая во внимание недостаточную популярность промышленного Ethernet для автоматизации, приближаются к своему виртуальному концу жизненного цикла для новых разработок, и по этой причине я отбросил все мысли о написании книги о них.

Arduino с новастал очевидным выбором из-за его огромной популярности, высокого уровня удобства для пользователя и, в конечном итоге, низкой стоимости. Программирование и реализация с использованием протоколов SAE J1939 с использованием Arduino требует, конечно, не только некоторых знаний в области программирования на C/C++, но и некоторых базовых знаний о технологиях Controller Area Network (CAN) и протоколе SAE J1939. Хотя я и пытаюсь объяснить подробности там, где это необходимо (т. е. где информация относится к программированию кода), полное подробное описание любой темы выходит за рамки этой книги.

Более подробную информацию о локальной сети контроллеров и SAE J1939 с м. в моих книгах :

- Понятное руководство по локальной сети контроллеров
- Понятный путеводитель по J1939

Оба издания доступны в мягкой обложке на Amazon.com (включая все их языковые сайты), Barnes & Noble (bn.com), Abebooks.com (включая все их языковые сайты) и в других книжных онлайн-магазинах по всему миру.

В том же смысле я также не буду заниматься объясением нового аппаратного обеспечения Arduino и его программирования. На эту тему достаточно множества книг по Arduino, Arduino Sketches и Arduino Shields, и нет смысла повторять информацию в них только для того, чтобы увеличить количество страниц.

Я, однако, кратко упомяну аппаратное обеспечение, используемое в моих проектах J1939, а именно Arduino Uno, Arduino Mega 2560 и CAN Shield, но только с целью представления информации, которая позволит читателю воспроизвести мои проекты. Я считаю также важным указать на ограничения производительности оборудования. В конце концов, Arduino может быть идеальным решением для прототипирования, но внедрение, например, полноценного протокола SAE J1939 определенно означало расширение границ.

Я также обращаюсь к некоторым темам программирования, которые не только входят в рамки нового Arduino, но и отражают мой личный подход к написанию читаемого кода. Эти темы включают стиль программирования, отладку кода и управление памятью.

ARD1939 — стек протоколов SAE J1939 для Arduino

Я считаю необходимым добавить несколько нетехнических (и, возможно, политически некорректных) аспектов разработки наиболее интересной функции этой книги — ARD1939, стека протоколов SAE J1939 для Arduino.

Внедрение стека протоколов SAE J1939 было (и во многих случаях остается) недоступным для многих инженеров. Программное обеспечение (т.е. исходный код) либо слишком завышено по цене, либо продаётся с большими лицензионными отчислениями (объектный код, библиотеки), что означает, что вы должны платить за каждую копию. По крайней мере, в случае с Arduino.

оборудование, это изменится, и проект с текущими протоколами ARD1939 доступен для бесплатной загрузки.

Честно говоря, разработка протокола J1939 не предстает большого труда для опытного программиста, единственное препятствие состоит в том, что вам придется потратить хорошие деньги на официальный документ SAE J1939 Standards Collection. Я считаю, что мой код также хорош, как и любые коммерческие доступные текущие протоколы. И да, я успешно протестировал свой код на нескольких имеющихся в продаже устройствах J1939.

Я обирался выпустить ARD1939 в виде оригинального исходного кода, но в конечном итоге отказался от этого, в основном из уважения к тем малым предприятиям, которые зарабатывают на жизнь продажей устройств и программных инструментов SAE J1939. Вместо этого я предоставляю предварительно скомпилированный код.

Исходный исходный код может быть легко адаптирован к любому другому встроенному оборудованию (у меня оно уже работает в системе ARM) и даже к ПК с Windows или Linux с возможностью CAN.

Кроме того, стоимость готового промышленного оборудования с расширенным температурным диапазоном в наши дни удивление низкая. По моему опыту, вы можете легко купить узел SAE J1939 с портом USB и Ethernet (приложение шлюза) с корпусом промышленной прочности менее чем за 200 долларов США при объеме = 1.

Я считаю, что средний пользователь Arduino, используя предварительно скомпилированный код ARD1939, вполне способен писать эффективные приложения J1939, независимо от того, есть ли у него доступ к исходному коду. В конце концов, нет никакой реальной необходимости модифицировать работающий код. Он поддерживает все функции протокола SAE J1939, и основное внимание всегда должно уделяться реальному приложению.

об авторе

Я являюсь автором серии технической литературы «Понятие руководство», охватывающей такие темы, как есть контроллеров (CAN), SAE J1939, промышленный Ethernet и определение размеров серводвигателя. Я работаю в отрасли CAN с 1997 года, а до этого был инженером по управлению движением в бумажной промышленности. У меня есть степень магистра электротехники, полученная в Университете Буппераля в Германии.

За последние годы я провел множество семинаров по промышленным системам полевых шин, таким как CAN, CANopen, SAE J1939, Industrial Ethernet и многими другим, во время различных конференций по всем траиваемым вычислительным системам реального времени (RTECC), ISA (Общество приборостроения, с системами автоматизации), конференции и различные другие мероприятия по всей территории Соединенных Штатов и Канады.

У меня была возможность много путешествовать по миру, но в 1989 году я поселился в Новой Англии. В настоящее время я живу в старом фермерском доме в Гринфилде, штат Массачусетс, с своей рыжеволосой зеленоглазой женой ирландско-американской происхождения и нашим сыном Патриком.

Для получения дополнительной информации о моих работах и для связи со мной посетите мой веб-сайт <http://copperhilltech.com>.

Связаться с автором

Несмотря на все усилия по подготовке этой книги, всегда существует вероятность того, что некоторые аспекты или факты не найдут свое общего одобрения, что побуждает нас, автора и издателя, просить ваше мнение. Если вы хотите предложить какие-либо поправки или исправления, пожалуйста, отправьте нам свой комментарий. Мы рассматриваем любую поддержку в дополнении этой книги, и мы приветствуем все обсуждения, которые способствуют тому, чтобы тема этой книги была максимально полной и объективной.

Чтобы предстатьвить поправки и исправления, войдите на сайт автора по адресу <http://copperhilltech.com/contact-us/> и оставьте примечание.

Кроме того, зарегистрируйтесь как пользователь на <http://www.j1939forum.com>, где мы создали раздел для авторов. Пожалуйста, используйте этот форум для связи с автором. Однако используйте его только для запроса информации, относящейся к содержанию этой книги. Любые запросы о помощи для конкретных пользовательских проектов и/или отладки будут игнорироваться.

Скачать код

Каждый пример программирования в этой книге, будь то скрипты Arduino C/C++ или проекты Visual Studio C#, доступны для бесплатной загрузки. Это также включает проект с текстом протоколов ARD1939.

Чтобы загрузить скетчи, проекты и библиотеки, перейдите по ссылке:

<http://ard1939.com>

Я подумал о том, чтобы перечислить все проекты Arduino и их код в печати в конце этой книги, и не для того, чтобы увеличить количество страниц, а из уважения к тем, кто не может загрузить код (да, они существуют). Однако все скетчи Arduino используют библиотеку MPC2515 от Кори Фаулера (см. главу «Программирование Arduino (эскизы)»), которую я не мог представить в печатном виде из-за ограничений авторского права. Как следствие, для полноценного использования кода, представленного в этой книге, вам необходимо подключение к Интернету для загрузки.

Все эскизы Arduino и другие образцы кода и проекты, представленные в этой книге, являются с свободным программным обеспечением; вы можете распространять и/или изменять их. Программы внесяются в расчете на то, что они будут полезны, но БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ; даже без подтверждения моей гарантии КОММЕРЧЕСКОЙ ПРИГODНОСТИ или ПРИГODНОСТИ ДЛЯ ОПРЕДЕЛЕННОЙ ЦЕЛИ. Загружая эти программы, вы подтверждаете, что эти примеры кода и проекты были созданы только для демонстрационных и образовательных целей.

1. Введение

Любой проект встроенных вычислений касается как аппаратного обес печения , так и прог раммног о обес печения . В этом конкретном проекте конечной ц елью я вля етс я загус к с тека протоколов SAE J1939 (прог раммное обесечение) наArduino (аппаратное обесечение).

Всег о не сколько лет назад такой проект требовал значительных инвестиц ий во встроеннное оборудование в с очетании с крутым обучением, но, как и во мног их других областях встроенног о прог раммирования , с емейство микронтроллеров Arduino значительно упрос тило с оздание даже с ложных встроенных решений.

Более того, наличие различных плат расширения (щитов), таких как щит CAN, открыло дверь в бесконечные возможнос ти.

В том же смысле ис пользование с тека протоколов SAE J1939 было финансово недоступно для мног их инженеров. Прог раммное обесечение (исх одный код) либо с ильно завышено, либо поставля етс я с лицензионными отчислениями (объектный код, библиотеки), что означает, что вы должны платить за каждую копию . По крайней мере, в случае с оборудованием Arduino это изменится . Проект с тека протоколов ARD1939 доступен для бесплатной загрузки.

Однако, прежде ч ем мы углубимс я в прог раммирование с тека протоколов SAE J1939, нам не обходимо охватить некоторые ос новы, такие как аппаратное обесечение Arduino, некоторые специальные темы прог раммирования , протокол CAN (локальная сеть контроллеров) и, естественно, стандарт SAE J1939.

1.1 Arduino Uno и Mega 2560 глю с CAN Shield

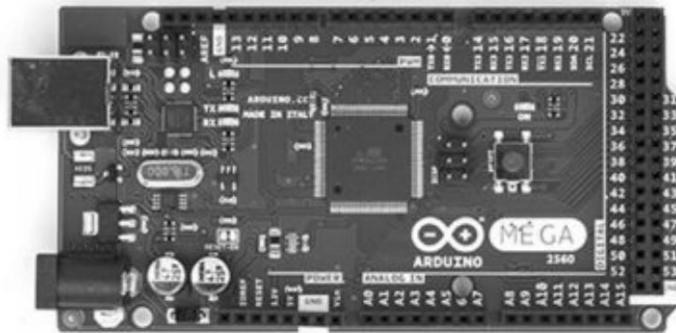
Как я упоминал ранее в этой книге, предполагается, что у вас есть некоторые базовые знания об Arduino Uno и самой Mega 2560, этих изах Arduino и Arduino Shields.



Чтобы разработать и протестировать примеры программ (этих изов), как показано в этой книге, я сначала исользовал Arduino Uno. Аппаратное обеспечение состоит из аппаратной платы с открытым исходным кодом, обычно разработанной на основе 8-битного микроконтроллера Atmel AVR с 2 КБ ОЗУ (рабочая память), 32 КБ флэш-памяти (этих изов) и 1 КБ EEPROM (энергонезависимой).

Этих технических спецификаций более чем достаточно для базового прототипирования приложений CAN и J1939 для проверки концепции. Однако (и я повторю этот момент) с простыми требованиями к скорости выполнения и расширенной функциональностью Arduino Uno может быть превышать с их пределов, особенно из-за ограниченной рабочей памяти в 2 КБ.

Чтобы преодолеть ограничения памяти Uno (см. также мои заметки об ограничениях производительности с ледяной шеей грави), я также исользовал Arduino Mega 2560, что, в свою очередь, позволило мне запустить полный стек протоколов SAE J1939 на платформе Arduino.



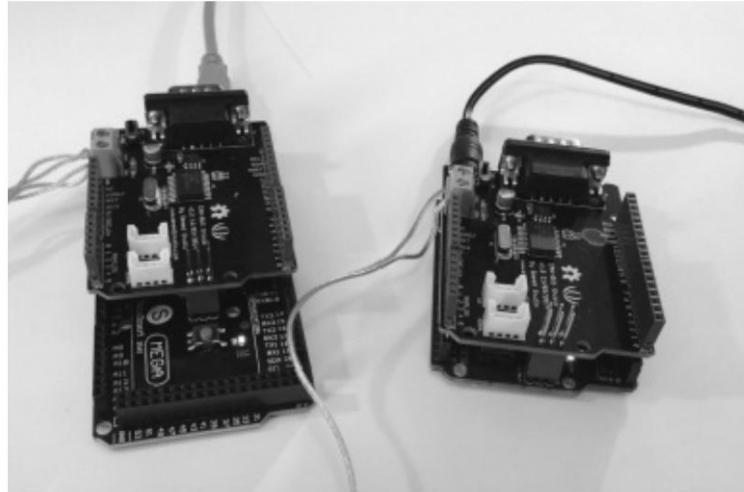
Arduino Mega 2560 — это плата микроконтроллера на базе процессора ATmega2560. По сравнению с Arduino Uno, он поставляется (помимо множества других улучшений) с ощутительно увеличенным объемом памяти, в частности, с 8 КБ SRAM по сравнению с 2 КБ Uno. Кроме того, он обеспечивает 256 КБ флэш-памяти (если) и 4 КБ EEPROM (энергонезависимой).

С точки зрения программирования обе версии, Uno и Mega 2560, совместимы.

Примечание. Все программы Arduino (если), показанные в этой книге, были разработаны и протестированы с Arduino Uno и Mega 2560. Нет никакой гарантии, что эти программы будут работать «как есть» на любой другой совместной системе.

Это кажется очевидным, но многие энтузиасты электроники изначально не понимают, что для сети требуется как минимум два узла; иначе вы не сможете установить связь. Если у вас нет реальной автомобильной сети (дизельный двигатель или автомобиль), достаточно для тестирования, вам нужно будет купить второй Arduino с защитным экраном CAN или приобрести имеющийся в продаже шлюз CAN с интерфейсом Windows (например, программное обеспечение для анализа и мониторинга CAN).).

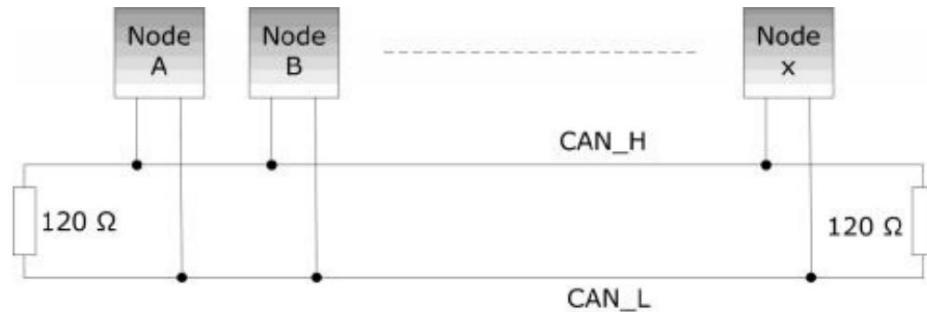
На следующем изображении показана конфигурация моей сети с Arduino Uno и Mega 2560 (на самом деле у меня есть еще несколько узлов CAN/J1939, чтобы моделировать полную сеть автомобиля).



Для разводки сети я использую обычные телефонные или акустические провода, что вполне приемлемо в лабораторных условиях. Просьба имейте в виду, что этот вид проводки может вызывать проблемы при подключении к реальному трансポートному средству.

Для «реальных» кабелей я рекомендую поискать в Интернете кабели SAE J1939 или ODB-II.

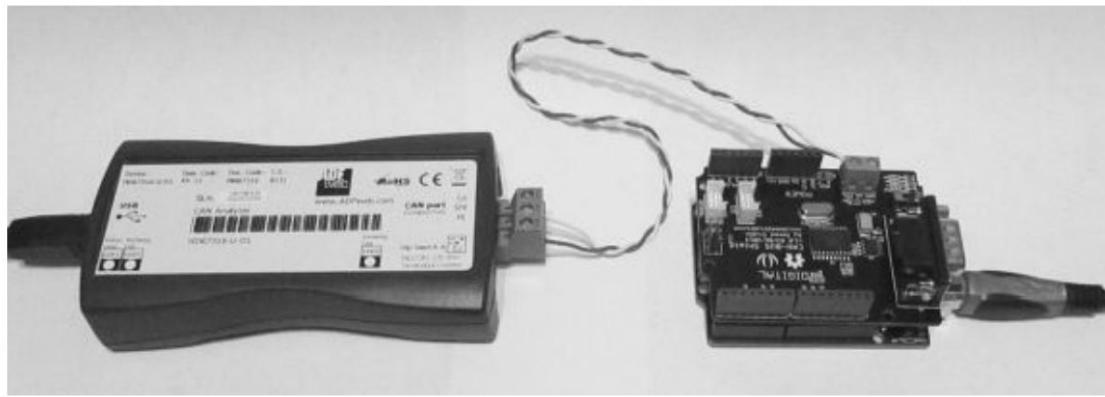
Кроме того, чтобы обеспечить правильную работу, вы должны знать, что в вашей сети должны использоваться симметричные резисторы, как показано на следующем рисунке (см. также главу Введение в сеть контроллеров).



Эмпирическое правило заключается в использовании симметричных резисторов на каждом конце сети, т. е. не более двух резисторов. Экраны CAN, о которых я расскажу в одной из последующих глав, имеют встроенные симметричные резисторы, но их нельзя переключать.

Когда вы работаете с более чем двумя узлами CAN/J1939 (например, два Arduino с щитом CAN плюс шлюз CAN/J1939, как у меня), вы должны знать обстоятельстве, что у вас может быть более двух терминалов.

резисторы в сеть, что может привести к сбою в сети. В результате вам, возможно, придется удалить резисторы (см. информацию производителя о рас положении и/или удалении с отключением резисторов).



Чтобы протестировать и проверить правильность передачи и приема с общей CAN, я использую шлюз ADFweb CAN-to-USB с его интерфейсом Windows.

Примечание. Чтобы протестировать приложение CAN/J1939, вам потребуется как минимум два узла CAN/J1939 для установления сетевого соединения. Вторым узлом может быть другой Arduino с щилдом CAN или (если позволяет бюджет) другое устройство CAN/J1939 с возможностью мониторинга данных CAN/J1939.

1.1.1 Экран CAN

Для тех, кто еще не знаком с автомобильным протоколом SAE J1939 (но использует эту книгу, чтобы получить больше информации по теме), J1939 использует контроллеры локальной сети контроллеров (CAN) для физического соединения (см. также главу Краткое введение в протокол SAE J1939). Arduino Uno и Mega 2560 по умолчанию не поставляются с встроенным контроллером CAN, и нам нужно использовать щит CAN.

Поскольку есть контроллеров (CAN) вновь пред назначена для промышленных решений (в отличие от гораздо более популярного USB для непромышленного использования, например дома и в лаборатории), на рынке не так уж много вариантов.

Путем некоторых исследований (т.е. просмотра) я нашел два очень похожих решения, и оба они работают с одной и той же библиотекой CAN (как объясняется в следующей главе). Оба решения используют CAN-контроллер Microchip MCP2515. Кроме того, оба решения

распространяется через всемирные онлайн-ресурсы.

Примечание. Далее я привожу ссылки на продукты, которые я использовал при написании этой книги. Однако продукты и их возможности могли измениться с момента публикации этой книги. Настоятельно рекомендуется проверять веб-сайты производителей (или дистрибуторов и представителей) для получения обновленной информации.

1.1.1.1 CAN-контроллер Microchip MCP2515

MCP2515 компании Microchip Technology — это автономный контроллер локальной сети контроллеров (CAN), реализующий спецификацию CAN версии 2.0B. Он способен передавать и принимать как стандартные, так и расширенные данные и удаленные кадры. MCP2515 имеет две маски приема и шесть фильтров приема, которые используются для фильтрации нежелательных сообщений, тем самым снижая нагрузку на микроконтроллер (MCU). MCP2515 взаимодействует с микроконтроллерами (MCU) через стандартный последовательный периферийный интерфейс (SPI).

Функции включают в себя два буфера приема с приоритетным ранением сообщений, шесть 29-битных фильтров, две 29-битные маски и три буфера передачи с функциями приоритизации и прерывания. (Источник: техническое описание микрочипа)

Примечание. Спецификация CAN 2.0B относится к возможностям использования стандартных кадров CAN с 11-битным идентификатором сообщения, а также расширенного формата с 29-битным идентификатором сообщения (как требуется для J1939).

Чтобы загрузить полное техническое описание MCP2515, войдите на страницу: <http://ww1.microchip.com/downloads/en/DeviceDoc/21801G.pdf>.

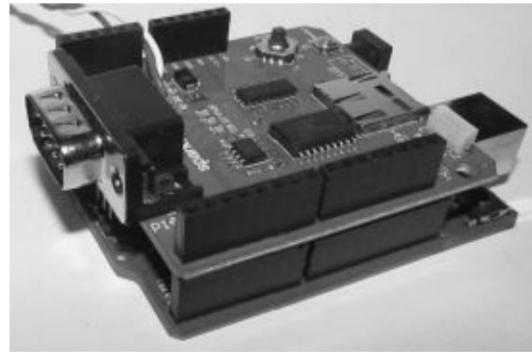
Оба экрана CAN, как описано в следующих главах, также используют приемопередатчик CAN Microchip MCP2551, который преобразует внутренние сигналы TTL в дифференциальное напряжение в соответствии с требованиями стандартов CAN.

Чтобы загрузить полное техническое описание MCP2551, войдите на страницу: <http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf>.

Примечание. В то время как MCP2515 предоставляет фильтры и маски с сообщениями, их программирование предоставляет собой проблему, граничащую с практической бесполезностью (исходный код предоставленный Microchip, по умолчанию бесполезен). MCP2515

Код драйвера, ис пользованный в проектах Arduino в этой книге, действительно относится к фильтрации с сообщений, но автор кода не гарантирует правильную функциональность. По этой причине вся фильтрация с сообщений в представленных проектах Arduino выполняется ядром.

1.1.1.2 Arduino CAN-Bus Shield от SK Pang electronics



Этот щит от SK Pang electronics обеспечивает возможность Arduino CAN-Bus.

Как было сказано ранее, он использует CAN-контроллер Microchip MCP2515 с CAN-трансивером MCP2551. Соединение CAN реализовано через стандартный 9-контактный разъем D, однако назначение контактов для CAN_H, CAN_L не соответствует стандарту.

Примечание. По правде говоря, не существует обязательного стандарта для назначения контактов, но в промышленности качестве виртуального стандарта используются контакты 2 (CAN_L) и 7 (CAN_H).

Я рекомендую использовать бортовые контакты CAN_L и CAN_H, чтобы припаять кабель CAN непосредственно к плате.

Задний экран также поставляется с держателем карты USB, последовательным разъемом для ЖК-дисплея и разъемом для модуля GPS EM406, что делает его пригодным для приложений реального времени данных.

Функции

- CAN v2.0B до 1 Мбит/с
- Высокоскоростной интерфейс SPI (10 МГц)
- Стандартные и расширенные данные и удаленные кадры

- Подключение по CAN через стандартный 9-контактный разъем sub-D.
- Как вариант, питание на Arduino может подаваться через sub-D через с амовос с танавливающими я предох ранитель и защиту от обратной полярности.
- Гнездо для GPS-модуля EM406 Держатель
- карты Micro SD Разъем для
- последовательного ЖК-дисплея
- Кнопка сброса
- Джойстик Управление навигацией меню Два
- с светодиодных индикатора

Примечания

- Нет кабелей в комплекте
- Шифты не включены; их нужно заказывать отдельно
- Назначение контактов для CAN_H, CAN_L не соответствует стандарту

Всю техническую информацию по использованию CAN-контроллера, держателя карты USB, джойстика, с светодиодов и т. д. можно найти на вики-сайте компании по адресу: <https://code.google.com/p/skpang/>

Информация для заказа

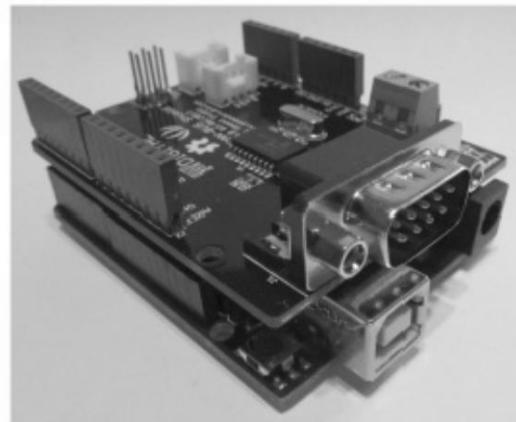
Чтобы заказать щит CAN для электроники SK Pang, вы можете использовать следующие ресурсы (или найти «Arduino CAN-BUS Shield» для получения дополнительных вариантов):

Спаркфан - <https://www.sparkfun.com/products/10039>

Электроника SK Pang - <http://skpang.co.uk/catalog/arduino-canbus-shield-with usd-card-holder-p-706.html>

Примечание. Версия щита SK Pang CAN, которую я использовал для этой книги, отлично работала с Arduino Uno R3, но не работала с Arduino Mega 2560, но у меня не было времени, ни ресурсов, чтобы точно определить проблему. Вполне может быть, что более новые версии щита будут работать. Рекомендуется уточнить у производителя совместимость с версией Arduino, отличными от Uno.

1.1.1.3 CAN-BUS Shield от Seeed Studio



По возможностям CAN шилд от Seeek Studio обладает той же функциональностью, что и шилд от SK Pang electronics, однако имеет гораздо меньшую цену, поскольку не имеет никаких дополнительных компонентов, кроме CAN-интерфейса.

Примечание. Во время написания этой книги я предпочитал шилд Seeed Studio CAN Bus из-за его совместимости между двумя системами, которые я в основном использовал, Arduino Uno и Arduino Mega 2560. Мега с его увеличенными ресурсами памяти позволила мне загрузить полный SAE. Стек протоколов J1939 на платформе Arduino, что невозможно с Uno.

В целом устройство производит солидное впечатление, тем более, что соединение CAN соответствует стандарту и, кроме того, обеспечивает подключение CAN через легкодоступные клеммы.

Функции

- Реализует CAN V2.0B с скоростью до 1 Мбит/с
- Интерфейс SPI до 10 МГц
- Стандартные (11 бит) и расширенные (29 бит) данные и удаленные кадры
- Два приемных буфера с приоритетным ранением сообщений
- Промышленный стандартный 9-контактный разъем Sub-D
- Два светодиода индикаторы

Примечания

- Нет кабелей в комплекте

Всю техническую информацию об использовании CAN-контроллера можно найти на вики-сайте компании по адресу http://www.seeedstudio.com/wiki/CAN_BUS_Shield.

Информация для заказа

Чтобы заказать щилд Seeed Studio CAN, вы можете использовать следующие ресурсы (или просмотреть «Arduino CAN-BUS Shield» для получения дополнительных опций):

Студия Seeed — <http://www.seeedstudio.com/depot/CANBUS-shield-p-2256.html>

Примечание. Защитный экран CAN-шины Seeed Studio претерпел некоторые аппаратные изменения, чтобы снять с овместимым с такими системами, как Arduino Mega 2560.

Версия 1.0 будет работать с Arduino Uno, а все более поздние версии также работают с Mega 2560. Это также повлияет на код проектов Arduino, в частности на строку «MCP_CAN CAN0(10);». в основном модуле, выбрав контакт CS. Эта строка должна измениться на «MCP_CAN CAN0(9);». для всех версий щиты CAN выше 1.0. Я добавил комментарий в соответствующий раздел кода.

1.1.2 Ограничения производительности

Важно знать, что ни Arduino Uno, ни Arduino Mega 2560, несмотря на то, что они идеально подходят для прототипирования из-за низкой цены и простоты программирования, в чистом виде не являются промышленным решением не только с точки зрения защиты окружающей среды. с пещириками (например, диапазон температуры и т. д.), но также и в отношении сокращения выполнения и, в частности, с Uno, ресурсов памяти.

Например, когда дело доходит до приложений CAN с окоростью 1 Мбит/с и большим трафиком данных, Arduino Uno быстро достигает своих пределов. Кроме того, обе версии, Uno и Mega 2560, в их нынешнем виде не подходят для полноценного использования CAN-USB из-за ограничений сокращения встроенного интерфейса USB (см. также с ледущую главу).

Однако картина меняется, когда речь идет о приложениях SAE J1939, для которых требуется только сокращение передачи 250 Кбит/с. В результате Arduino Uno

хорошо подходит для приложений мониторинга и моделирования SAE J1939 в отношении с коротким выполнением.

Однако есть ограничения на объем памяти, особенно разреженные 2 КБ SRAM, делают, к сожалению, невозможным реализацию и запуск полнофункционального протоколов J1939. Тем не менее, я предстаю решение (то есть ограниченную версию стека протоколов ARD1939) в последующих главах.

Проблема предстаёт на транспортном протоколе (TP) в соответствии с SAE J1939/21, управляющим передачей кадров данных, превышающими стандартный 8-байтовый формат CAN. TP поддерживает до 1785 байт на сообщение, и вам потребуется как минимум вдвое больший размер буфера, поскольку спецификация SAE требует, чтобы управляющее приложение (CA) J1939 поддерживало два сервиса (сервис BFM и сервис RTS/CTS) одновременно. Даже с обширной настройкой кода, например, запуском только одного параллельного сервиса, нет решения, которое позволило бы вам использовать эти 1785 байт, а оставшиеся 263 байта легли бы черпаться.

Тем не менее, Arduino Uno очень хорошо подходит для реализации без поддержки TP или с ограниченной поддержкой. В случае стека протоколов ARD1939 (который будет представлен в следующей главе) максимальная длина кадра данных ограничена 256 байтами, что все еще более чем полезно для простых целей тестирования и моделирования.

Когда дело доходит до простого мониторинга трафика данных транспортного средства J1939 (для которого не требуется реализация стека протоколов J1939), Arduino Uno подходит также, как и любая другая мощная процессорная система.

Чтобы поддерживать полную реализацию стека протоколов SAE J1939, я использовал Arduino Mega 2560 из-за его близкого совместимости с Arduino Uno, в частности, рабочего напряжения 5 вольт. Он предоставляет собой еще одно недорогое решение, но с значительно увеличенными объемами памяти (8 КБ SRAM вместо 2 КБ).

Другие системы, такие как Arduino Due, обеспечивают лучшую производительность процессоров с значительно увеличенными объемами памяти примерно по той же цене, что и Mega 2560.

Однако мы также входим в первую зону совместимости Arduino Shield. Согласно веб-сайту Arduino (arduino.cc), плата Due совместима с всеми

Шилды Arduino, работающие от 3,3 В. Реальность такова, что подавляющее большинство шилдов в настоящее время работают на 5В.

Официальный сайт Arduino тоже не очень помогает. В нем только говорится, что Mega 2560 разработан для совместимости с большинством экранов, разработанных для Uno, что в конечном итоге является очень распространенным заявлением.

В моем случае оказалось, что изначально ни один из двух экранов CAN-шины (Seeed Studio и SK Pang) не распознавался Mega 2560 (с использованием того же программного обеспечения, которое работает на Uno).

Проблема с Arduino Mega 2560 и шилдом Seeed Studio CAN была вызвана несовместимым назначением выводов интерфейса SPI, которое Seeed Studio исправил в более новых версиях.

В случае с Due проблема заключается в рабочем напряжении 3,3 В, что запрещает использование шилдов, работающих от 5 В. Кроме того, у Due фактически есть собственный выделенный интерфейс CAN, что теоретически исключает использование дополнительного экрана CAN. Однако на момент написания этой статьи не было достаточно информации о интерфейсе (кода). Ситуацию усугубляет то, что в интерфейсе CAN отсутствует необходимый приемопередатчик CAN, что делает его бесполезным для любых приложений CAN, если только вы не купите (или не создадите) внешнюю коммутационную плату.

И да, есть еще много вариантов Arduino с лучшей производительностью и увеличенными ресурсами памяти, но изучение всех возможных конструкций оборудования и тестирование наиболее многообещающих сценариев может оказаться дорогос团委м и трудоемким проектом.

В последующих главах я представлю несколько вариантов того, как использовать Arduino Uno и Arduino Mega 2560 для приложений SAE J1939:

1. Примеры моделирования и мониторинга SAE J1939. Эти примеры программирования предоставляются практически без ограничений в отношении их функциональности, и, пока образцы кода находятся в разумных пределах, не должно быть никаких технических проблем с точки зрения объема памяти.
2. Простой шлюз SAE J1939 для ПК: это приложение будет работать в обеих версиях, Uno и Mega2560, поскольку оно не требует реализации стандартных протоколов на Arduino. Приложение

прос то передает 29-битные с ообщения CAN между Arduino и ПК.

3. ARD1939 — встроенный стек протоколов J1939 для Arduino: ARD1939 поддерживает полный стек протоколов SAE J1939, включая процедуру утверждения адреса (SAE J1939/81) и транспортный протокол (TP, SAE J1939/21), однако в случае Arduino Uno максимальная длина сообщения составляет (из-за проблем с объемом памяти, как объяснялось ранее) 256 байт. Весьма для Arduino Mega 2560 составляетется без ограничений по длине сообщения, т.е. она поддерживает стандарт до 1785 байт на сообщение.

CAN Последовательный интерфейс и синхронизация 1.1.3 Собщения

Аппаратная конфигурация, предложенная в этой книге, а именно плата Arduino и щит CAN, предполагает, что она может хорошо подойти для шлюза CAN-USB или J1939-to-USB. Тем не менее, скорость передачи данных USB на Arduino ограничена 115 200 бод, что составляет примерно половину скорости соединения J1939 с скоростью 250 Кбит/с и примерно 11% скорости полноценного приложения CAN с скоростью 1 Мбит/с.

Надежное применение шлюза с Arduino возможно только при максимальной скорости передачи данных CAN не выше 100 Кбит/с, в противном случае существует риск потери сообщений.

Все это приводит к тому, что приложение должно перехватывать каждый кадр данных в сети. Одним из способов решения проблемы синхронизации является использование фильтров сообщений, т.е. приложение допускает только набор специфических для приложения идентификаторов сообщений или PGN соответственно.

Надежность приложения, использующего фильтры сообщений, сильно зависит от загрузки шины, т.е. от средней частоты сообщений CAN нашине.

При этом использование Arduino в качестве шлюза CAN-USB сопряжено с большой вероятностью потери сообщений. Даже использование больших буферов сообщений CAN не гарантирует получение всех сообщений.

С приложением J1939 ситуация иная, потому что:

- Стандарт SAE J1939 предписывает скорость передачи данных 250 Кбит/с.
- Типичное приложение J1939 будет использовать только небольшое подмножество всех датчика PGN, что делает использование фильтров с общими логическими выбором.
- Самая высокая частота сообщений в стандарте SAE J1939/71 составляет 10 миллисекунд.
- Загрузка шины в 60 % считается чрезвычайно высокой (100 % будет непрерывным потоком данных).

Таким образом, можно создать полноценный шлюз J1939-to-USB с помощью Arduino, при условии, что будут применены некоторые функции, такие как эффективная буферизация сообщений. Другой мерой может быть создание подпрограмм обслуживания прерываний для получения данных CAN и USB.

Другим аспектом является протокол связи между USB-интерфейсом Arduino и системой, которая получает данные USB. Ведь в таком шлюзовом приложении данные CAN преобразуются в сообщения USB (и наоборот) и эти сообщения должны иметь определенный формат. Протокол описывает преобразование этих сообщений.

Чем больше информации упаковано в протокол (например, байты заполнения и контрольная сумма), тем длиннее будет сообщение и тем больше времени потребуется для его передачи.

Среднее время передачи сообщения J1939 составляет примерно 540 микросекунд. USB-порту Arduino потребуется примерно 700 микросекунд для передачи того же сообщения в лучшем случае (за счет удаления служебных данных протокола CAN и идентификации любых функций безопасности, таких как контрольная сумма). Любые дополнения к протоколу (= служебные данные протокола) приведут к увеличению времени передачи в диапазоне от 1000 до 2000 микросекунд.

Хотя все это приводит к замедлению передачи кадров данных примерно в 1,4–4 раза, Arduino по-прежнему подходит для базового шлюза.

Примечание. Промышленные шлюзы CAN-to-USB или J1939-to-USB с их более высокой скоростью передачи USB в диапазоне Мбит/с используются в расширенных протоколах, которые включают, например, метки времени CAN-кадра.

1.1.4 Подключение Arduino к реальной сети SAE J1939

Я уже давно пробовал соединить Arduino на имеющихся в продаже устройствах SAE J1939 с их программным обеспечением для мониторинга и моделирования, но у меня не было такой необходимости, как подключение их к реальному транспортному средству (дизельному двигателю). Тем не менее, все скетчи должны работать в «реальном мире».

Если вы это сделаете, пожалуйста, внимательно прочтите описание всех проектов Arduino. Все эскизы были созданы для демонстрационных и образовательных целей, но их влияние на реальную автомобильную сеть сильно зависит от приложения, которое вы пытаетесь реализовать. Пристойный мониторинг фреймов данных J1939 не должен вызвать никаких проблем, но запись данных в сеть требует детального знания сети и ее компонентов. Я попытался рассмотреть любой такойспектр в отдельных проектах.

Чтобы подключить Arduino к реальной автомобильной сети, вам необходимо обес печить соответствующую проводку и разъемы. Далее я объясню схему в том виде, в каком они используются в промышленности, но мое описание может служить лишь грубым учебным примером. Дополнительную информацию см. в сборнике стандартов SAE J1939 (документы SAE J1939/1x и SAE J1939/2x).

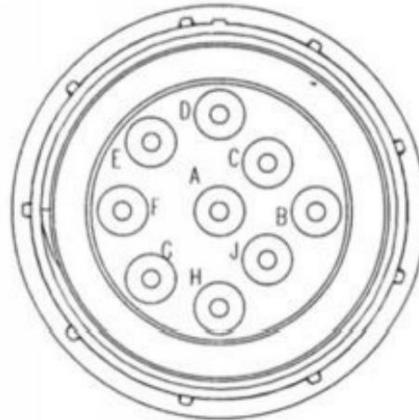
1.1.4.1 Внешний диагностический разъем SAE J1939/13

J1939/13 определяет стандартный разъем для диагностических целей. Он разрешает доступ к каналам связи автомобиля. Разъем - Deutsch HD10-9-1939 (9 контактов, круглый разъем).



Согласно официальному документу SAE J1939/13 Off-Board Diagnostics Connector, разъем поддерживает как носители с экранированной витой парой (как определено в SAE J1939/11), так и неэкранированные четырехжильные носители с витой парой (как определено в ISO 11783-2). Обозначения отдельных сигнальных проводов соответствуют стандарту CAN CAN_H и CAN_L. Для SAE J1939/11 третью соединение для подключения экрана обозначается CAN_SHLD.

Назначение контактов следующее:



Контакт А — аккумулятор (-)

Контакт В — аккумулятор (+)

Контакт С — CAN_H

Контакт D — CAN_L

Контакт E — CAN_SHLD

Контакт F — SAE J1708 (+)

Контакт G — SAE J1708 (-)

Контакт H — собственная шина CAN_H для использования или реализации OEM

Контакт J — собственная шина CAN_L для использования или реализации OEM

Для получения более подробной информации о разъеме и его подключении обратитесь к официальному документу SAE.

1.2 Программирование Arduino (наброски)

Реализация любого из представленных штотов CAN-BUS и соответствующих эквивалентов CAN прошла на удивление гладко в сочетании с правильным программным обеспечением библиотеки.

Примечание. Все скетчи Arduino CAN/J1939, описанные ниже, включают библиотеку MCP2515, а также интерфейс между слоями CAN и SAE J1939. Если вам не терпится приступить к программированию J1939, вы можете пропустить эту главу и использовать информацию, когда начнете свои собственные проекты CAN/J1939.

Я нашел несколько примеров исходного кода для доступа к CAN-контроллеру MCP2515, но большинство из них даже не прошли этап начального контроля качества (я сначала прочитал код, прежде чем использовать его). Одним из критериев качества была поддержка 29-битных идентификаторов с общим CAN (совместимость с CAN 2.0B), что является обязательным, когда речь идет о реализации автомобильного сетевого протокола SAE J1939. Некоторые образцы программного обеспечения, которые я нашел, были буквально «образцами», и они оставляли достаточно места для игры в угадайку.

Меня больше всего порадовала библиотека MCP2515 от Cory Fowler, которую можно найти по адресу https://github.com/coryjfowler/MCP2515_lib.

Эта библиотека совместима с любым экраном или интерфейсом CAN, используя один контроллер протокола CAN MCP2515.

1.2.1 Библиотека MCP2515

Как и в случае с любым последовательным сетевым контроллером, основными вызовами функций являются:

1. Инициализация
2. Чтение данных
3. Запись данных
4. Проверка состояния

В случае библиотеки MCP2515 эти функции предстают:

1. Инициализация : CAN0.begin

2. Чтение данных : CAN0.readMsgBuf
вкл. CAN0.checkReceive, CAN0.getId 3. Данные
записи: CAN0.sendMsgBuf 4. Статус
проверки: CAN0.checkError

1.2.1.1 Вызовы функций

Функция :	CAN0.begin
Цель:	Инициализирует контроллер CAN и устанавливает скорость (в бодах).
Параметр:	CAN_5KPS ... CAN_1000KPS (см.
Код возврата:	CAN_OK = Инициализация в порядке CAN_FAILINIT = Ошибка инициализации
Функция :	CAN0.checkReceive
Цель:	Проверить, было ли получено
Параметр:	
Код возврата:	с сообщением Нет CAN_MSGAVAIL = С сообщением доступно CAN_NOMSG = Нет с сообщениями
Функция :	CAN0.readMsgBuf Чтение
Цель:	буфера с сообщений. nMsgLen
Параметр:	возвращает длину с сообщениями (количество
байты данных)	nMsgBuffer возвращает фактическое с сообщение
Код возврата:	Никто
Функция :	CAN0.getId
Цель:	Извлекает идентификатор полученного сообщения .
Параметр:	Нет
Код возврата:	m_nID = идентификатор сообщения
Функция :	CAN0.sendMsgBuf
Цель:	Отправить буфер с сообщениями
Параметр:	id = идентификатор сообщения ext = CAN_STDID (11-битный идентификатор) или CAN_EXTID (29- идентификатор бита)

len = количество байтов данных (0...8)

buf = буфер с сообщений

Код возврата: Нет

Функция: CAN0.checkError

Цель: Проверяет контроллер CAN на наличие ошибок

Параметр: Никто

Код возврата: CAN_OK = Статус в порядке

CAN_CTRLERROR = Ошибка

Существуют дополнительные функции, среди прочего, для фильтрации с сообщений и масок настроек, и их стоит проверить для более сложных функций, но они не нужны для простых задач связи CAN.

Реализация библиотеки MPC2515 довольно проста: откройте Arduino, создайте новый файл, затем используйте пункты меню Sketch->Add File..., чтобы включить в проект следующие файлы:

- mcp_can.cpp
- mcp_can.h
- mcp_can_dfs.h

В файле проекта Arduino с верху добавьте следующее:

```
#include "mcp_can.h"  
#include <SPI.h>  
  
MCP_CAN CAN0(10);
```

Позвольте мне повторить два важных момента, касающихся MCP2515:

1. Защитный экран CAN-шины Seeed Studio претерпел некоторые аппаратные изменения, чтобы сесть совместимым с такими темами, как Arduino Mega 2560. Версия 1.0 будет работать с Arduino Uno, в то время как все более поздние версии также работают с Mega 2560. Это также будет влиять на код проектов Arduino, в частности на строку «MCP_CAN CAN0(10);» в основном модуле, выбрав контакт CS.

Эта строка должна измениться на «MCP_CAN CAN0(9);» для всех версий щита шины CAN выше 1.0.

2. В то время как MCP2515 предоставляет фильтры и маски с сообщений, их программирование предоставляет собой проблему, граничащую с практической бесполезностью (исходный код, предоставленный Microchip, бесполезен по умолчанию). Код драйвера MCP2515, используемый в проектах Arduino в этой книге, действительно относится к фильтрации с сообщений, но автор кода не гарантирует правильную функциональность. Поэтому причине вся фильтрация с сообщений в предоставленных проектах Arduino выполняется программно.

1.2.1.2 Интерфейс CAN

Хотя код, предоставленный в предыдущей главе, был хорошо разработан и, следовательно, очень эффективен, я добавил еще один программный слой между интерфейсом CAN и текстом протоколов ARD1939.

Я написал исходный код ARD1939 на языке C (не C++), чтобы обеспечить высокий уровень совместимости с различными существующими системами и их компиляторами (да, код уже работает в системе ARM, и я опубликовал еще одну книгу об этом). Естественно, системы, отличные от Arduino Uno или Mega 2560, требуют других драйверов CAN, и было бы довольно обременительно найти и заменить все вызовы функций CAN в протоколе.

По этой причине я создал несколько «общих» вызовов функций, которые используются текстом протоколов. Для будущих реализаций мне нужно только переписать модуль can.cpp.

Примечание. В дополнение к вызовам функций MCP2515 я добавил кольцевой буфер с сообщений CAN. Буфер с сообщений MCP2515 ограничен, и при более высокой нагрузке на шину (высокая частота сообщений) вы будете терять сообщения.

Эти вызовы функций снова основаны на базовой структуре каждой программы последовательной связи:

1. Инициализация
2. Чтение данных
3. Запись данных
4. Проверка состояния

Эти функции представлены:

1. Инициализация : canInit()
2. Чтение данных : canReceive(...)
3. Запись данных : canTransmit(...)
4. Проверить статус : canCheckError()

Примечание. Для максимальной совместимости между компиляторами я воздержался от ис пользования структур и их указателей. Принцип KISS (Keep It Simple Stupid!) делает перенос легкой задачей и повышает читабельность кода. Все это идет с практическими ограничениями производительности.

1.2.2 Специальные темы программирования

У каждого программиста свой стиль кодирования, и я, безусловно, не исключение. Тем не менее, код всегда должен быть написан с учетом следующих аспектов:

- Высокий уровень удобочитаемости, в том числе множество комментариев (у вас возникнут проблемы с декодированием вашей собственной программы через несколько недель после того, как вы просматривали ее в последний раз)
- Простая отладка (здесь также помогает читабельность)
- Возможные ограничения памяти (особенно относится к Arduino Uno)
- Keep It Simple Stupid (нет смысле тратить драгоценное время на анализ/документирование с лишком ложных структур, массивов перечислений и т. д., если только они не дают больших преимуществ в отношении производительности)

С моим обычным стилем программирования я стараюсь следовать этим правилам, особенно в том аспекте, что я делаю с своим кодом с большим общественным. Если код недостаточно прост и интуитивно понятен, чтобы мои читатели могли понять его, не задавая вопросов, он обычно будет отклонен.

1.2.2.1 Венгерская нотация

Венгерская нотация — это соглашение об именах идентификаторов в компьютерном программировании, в котором имя переменной или функции указывает на ее тип или предполагаемое ис пользование. Венгерская нотация была разработана так, чтобы быть независимой от языка, и нашла свое первое серьезное применение в языке программирования BCPL. Поскольку в BCPL нет других типов данных, кроме машинного слова, в самом языке ничего не

помогает программисту запомнить типы переменных. Венгерская нотация призвана исправить это, предоставив я программисту ясное знание типа данных каждой переменной.

Венгерской нотации имя переменной начинается с группы строчных букв, которые представляют собой меморик для типа или назначения этой переменной, за которыми следует любое имя, выбранное программистом; эта последняя часть иногда выделяется как имя.

Первоначальная венгерская нотация, которая теперь будет называться Венгерской для приложений, была изобретена Чарльзом Симони, программистом, работавшим в Xerox PARC примерно в 1972–1981 годах, а затем ставшим главным архитектором Microsoft. Это могло быть получено из более раннего принципа использования первой буквы имени переменной для установки ее типа — например, переменные, имена которых начинаются с букв от I до N в FORTRAN, по умолчанию являются целыми числами. (Источник: Wikipedia.org)

В своем программировании, не следя «официальной» венгерской нотации до последней детали, я использую описательные имена переменных для улучшения читабельности кода, и им предшествуют одна или две строчные буквы, указывающие на тип переменной.

Например, я использую n для целого числа и s для строки. Просмотрите несколько примеров:

```
логический  
bCANDataReceived;  
байт cSerialData;  
int nj1939Статус; char sj1939Приложение[20];
```

Я воздержусь от предоставления «официального» списка соглашения об именах, потому что оно не существует. Однако, когда вы посмотрите на программы, которые я предоставляю, вы поймете идею (без необходимости адаптации стиля для ваших приложений). Мое простое намерение — предоставить информацию в образовательном стиле, основанную на отличной читабельности.

1.2.2.2 Отладка кода с помощью макросов

Хотя одной из сильных сторон Arduino Uno является просмотр программы встроенных решений, она может стать все более и более непривычной, когда дело доходит до отладки кода. Arduino IDE предоставляет очень ограниченные возможности отладки, если их вообще нет. Единственным решением проблемы является добавление кода, который преобразует переменную в строку и отображает ее на последовательном мониторе Arduino.

быть громоздким и трудоемким.

Другие, професиональные (и гораздо более дорогие) с реды программирования позволяют вам устанавливать точки останова и отображать значения переменных, даже массивов.

Пока не наступит время, когда Arduino IDE предоставит такие функции, давайте отладим наш код с помощью нескольких так называемых макросов C-программирования.

Программа C модифицирует файл исходного кода перед передачей его компилятору. Скорее всего, вы привыкли использовать программы C для включения файлов непосредственно в другие файлы или константы #define, но программы C также можно использовать для создания «встроенных» кода с использованием макросов, расширяемых во время компиляции, и для предотвращения повторной компиляции кода.

Другим важным применением программы является определение макросов. Преимущество макроса состоит в том, что он может быть нейтральным к типу (это также может быть недостатком, конечно), и он всегда транслируется непосредственно в код, поэтому нет никаких накладных расходов на вызов функции.

Источник: <http://www.cprogramming.com/tutorial/cpreprocessor.html>.

Используя макросы программы C, я разработал следующие инструменты для отладки приложений Arduino:

- DEBUG_INIT() — определяет строковую переменную для отладки. Эта функция обязательна для запуска отладки, и ее следует поместить в начало модуля кода (файла), который вы хотите отлаживать.
- DEBUG_PRINTEX(T, v) — печатает переменную в шестнадцатеричном формате с предшествующим текстом.
- DEBUG_PRINTDEC(T, v) — печатает переменную в десятичном формате с предшествующим текстом.
- DEBUG_PRINTARRAYHEX(T, a, l) — выводит массив переменных определенной длины в шестнадцатеричном формате с предшествующим текстом.
- DEBUG_PRINTARRAYDEC(T, a, l) — выводит массив переменных определенной длины в десятичном формате с предшествующим текстом.
- DEBUG_HALT() — останавливает программу до тех пор, пока пользователь не отправит нажатие клавиши через последовательный монитор.

Примечание. Макрос считывает по одному символу за раз, т. е. если вы отправляете более одного символа, программа будет продолжаться до тех пор, пока не будут прочитаны все символы. В некоторых случаях это может быть полезной функцией.

Переменные, используемые для макросов:

- Т – Текст, который будет отображаться с переменной (например, имя переменной)
- v – Фактическая переменная; может быть любого типа (например, int, byte, float и т. д.)
- a — указатель на массив переменных
- l — длина массива

Инструкции DEBUG, находящиеся в файле debug.h, являются частью всех проектов кода Arduino в этой книге (см. фактический код в приложении).

Результат отладочной печати отображается на последовательном мониторе Arduino.

Следующий код предоставляет общий пример программы, демонстрирующей использование макросы:

```
DEBUG_INIT()
```

```
// Вызов с текстом протокола
J1939 nj1939Status = j1939Operate(&nMsgID, &IPGN, &pMsg[0],
    &nMsgLen, &nDestAddr, &nSrcAddr, &nPriority);
```

```
DEBUG_PRINTHEX (<<nDestAddr = <<, nDestAddr)
DEBUG_PRINTDEC (<<nDestAddr = <<, nDestAddr)
```

```
DEBUG_PRINTARRAYHEX ("pMsg = ", pMsg, nMsgLen)
DEBUG_PRINTARRAYDEC("pMsg = ", pMsg, nMsgLen)
```

```
DEBUG_HALT()
```

2. Краткое введение в протокол SAE J1939

Подкомитет Общества автомобильных инженеров (SAE) по контролю и связи между грузовыми автомобилями и автобусами разработал ряд стандартов, касающихся проектирования и использования устройств, которые передают электронные сигналы и управляемую информацию между компонентами автомобиля. SAE J1939 и сопровождающие его документы были созданы общепринятым отраслевым стандартом и сеть транспортных средств, которые выбирают для внедорожных машин в таких областях, как строительство, погрузочно-разгрузочные работы, машины транспорта, лесозаготовительные машины, сельскохозяйственная техника, морское и военное применение.

Примечание. Справедливо сказать, что протокол транспортных средств SAE J1939 в основном используется для дизельных двигателей, который охватывает все ранее упомянутые области применения.

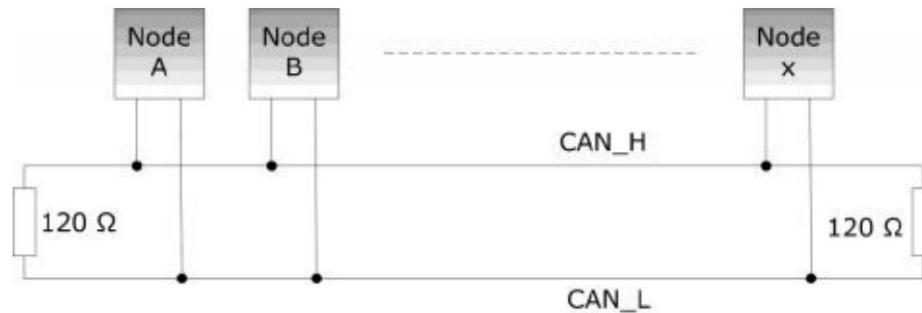
J1939 — это протокол более высокого уровня, основанный на локальной сети контроллеров (CAN). Он обеспечивает последовательную передачу данных между микропроцессорными системами (также называемыми электронными блоками управления - ECU) в любых транспортных средствах большой грузоподъемности. Сообщениями, которыми обмениваются эти блоки, могут быть такие данные, как скорость автомобиля, сообщение управления крутящим моментом от трансмисии к двигателю, температура масла и многое другое.

Примечание. Несмотря на то, что протокол SAE J1939 существует уже много лет, он по-прежнему набирает популярность, особенно с учетом более широкого использования систем управления автопарком, которым неизбежно потребуются данные из сети транспортных средств, например, для расчета циклов технического обслуживания... Управление автопарком также тесно связано с Интернетом вещей (IoT), а транспорт считается одним из самых быстрорастущих рынков для IoT.

2.1 Введение в сеть контроллеров

Сеть контроллеров (CAN) — это технология последовательной сети, которая изначально была разработана для автомобильной промышленности, особенно для европейских автомобилей, но также стала популярнойшиной в промышленной автоматизации, а также в других приложениях. Шина CAN в основном используется в встроенных системах, как следует из ее названия, представляя собой сетьевой телеграфию, которая обеспечивает быструю связь между микроконтроллерами в соответствии с требованиями реального времени, устраняя необходимости в градации более дорогой и сложной телеграфии двух портовой оперативной памяти.

CAN — это двухпроводная, полу duplexная, высокоскоростная сеть, которая намного превосходит традиционные последовательные технологии, такие как RS232, в отношении функциональности и надежности, и при этом реализация CAN более эффективна с точки зрения затрат.



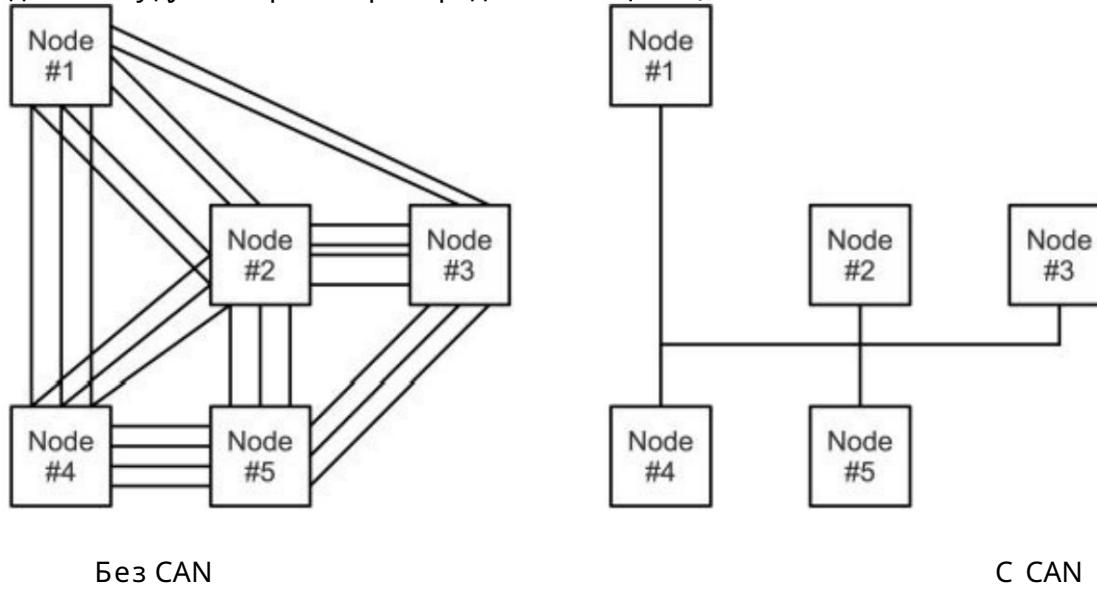
В то время как, например, TCP/IP предназначен для передачи больших объемов данных, CAN предназначен для работы в режиме реального времени и со скоростью передачи 1 Мбит/с может легко превзойти соединение TCP/IP со скоростью 100 Мбит/с. к короткому времени реакции, свое временному обнаружению ошибок, быстрому устранению ошибок и устранению ошибок.

Сети CAN можно использовать как встроенную систему связи для микроконтроллеров, а также как открытую систему связи для интеллектуальных устройств. Некоторые пользователи, например, в области медицинской техники, выбрали CAN, потому что они должны соответствовать особенностям требований безопасности.

Аналогичные требования должны учитываться производителями другого оборудования с очень высокими требованиями к безопасности или надежности (например, роботы, подъемники и транспортные системы).

Наибольшее преимущество локальной сети контроллеров заключается в уменьшении количества проводов в счетании с их рациональным предотвращением коллизий сообщений (т.).

данные будут потеряны при передаче с сообщения).



Ниже приведен краткий обзор технических характеристик CAN.

Локальная сеть контроллера

- Это последовательная сетьовая технология для встраиваемых решений.
- Требуется только два провода с именами CAN_H и CAN_L.
- Работает со скоростью передачи данных до 1 мегабита в секунду.
- Поддерживает максимум 8 байтов на кадр с сообщения.
- Не поддерживает идентификаторы узлов, только идентификаторы с сообщений. Одно приложение может поддерживать несколько идентификаторов с сообщений.
- Поддерживает приоритет с сообщения, т.е. чем меньше идентификатор с сообщения, тем выше его приоритет.
- Поддерживает две длины идентификатора с сообщения: 11-битную (стандартную) и 29-битную (расширенную).
- Не возникают конфликты с сообщений (поскольку они могут возникать при использовании других последовательных технологий).
- Не требователен к кабелю. Достаточно витой пары.

Примечание. Для получения более подробной информации о CAN см. официальную спецификацию CiA/Bosch или «Понятное руководство по локальной сети контроллеров», упомянутое в приложении к этой книге.

Machine Translated by Google

2.2 Протокол верхнего уровня SAE J1939

Несмотря на то, что CAN чрезвычайно эффективен в автомобилях и небольших встроенных приложениях, сам по себе CAN не подходит для проектов, требующих минимального сетевого управления и сообщений с более чем восемью байтами данных.

Как следствие, протоколы более высокого уровня (дополнительное программное обеспечение поверх физического уровня CAN), такие как SAE J1939 для транспортных средств, были разработаны для обеспечения усовершенствованной сетевой технологии, которая поддерживает сообщения неограниченной длины и позволяет управлять сетью, включая использование идентификаторов узлов. (CAN поддерживает только идентификаторы сообщений, когда один узел может управлять несколькими идентификаторами сообщений).

SAE J1939

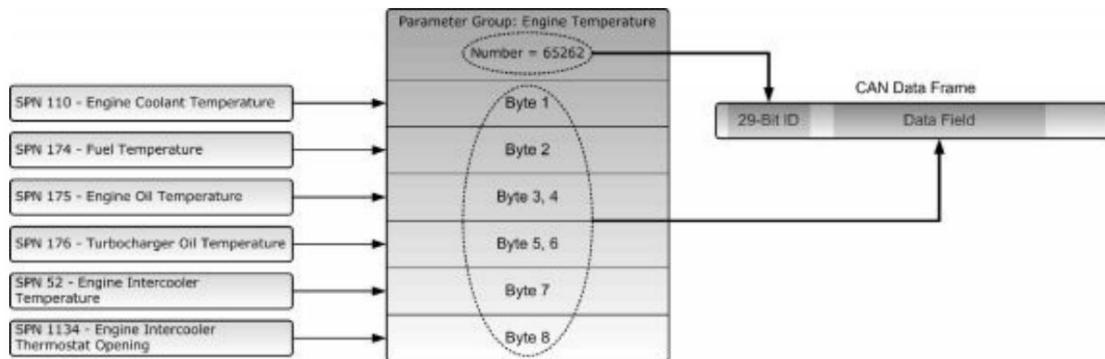
- Является стандартом, разработанным Обществом автомобильных инженеров (SAE).
- Определяет связь для автомобильных сетей (грузовиков, автобусов, сельскохозяйственной техники и т. д.)
- Является протоколом более высокого уровня, использующим CAN в качестве физического уровня. Использует экранированную витую пару. Максимальная длина сети составляет 40 метров (~120 футов).
- Применяется с стандартной скоростью передачи 250 кбит/с.
- Допускает до 30 узлов (ECU) в сети. Допускает до 253 приложений
- контроллера (CA), где один ECU может управлять несколькими СА. Поддерживает одноранговую и широковещательную связь. Поддерживает
- сообщения длиной до 1785 байт. Определяет набор номеров групп параметров (PGN, предопределенные параметры автомобиля)
-
- Поддерживает управление сетью (включая идентификаторы узлов и процессор запроса адреса)

Последнему с другими функционально-управляющими протоколами, такими как CANopen и DeviceNet, SAE J1939 в первую очередь управляется данными. На самом деле, J1939 обеспечивает гораздо лучшую пропускную способность, чем любой из этих протоколов автоматизации.

Пакеты данных J1939 содержат фактические данные и заголовок, который содержит индекс, называемый номером группы параметров (PGN). PGN определяет функцию сообщения.

и связанные с ними данные (более подробное описание PGN и основных функций протокола включено в описание стандартных протоколов ARD1939 далее в этой книге). J1939 пытается определить стандартные PGN, чтобы охватить широкий спектр автомобильных, сельскохозяйственных, морских и внедорожных транспортных средств.

Примечание. Для получения более подробной информации о борнике стандартов SAE J1939 см. официальную документацию SAE или «Полное руководство по J1939», упомянутое в приложении к этой книге.



Фактические данные в поле данных описываются именами участников служб.

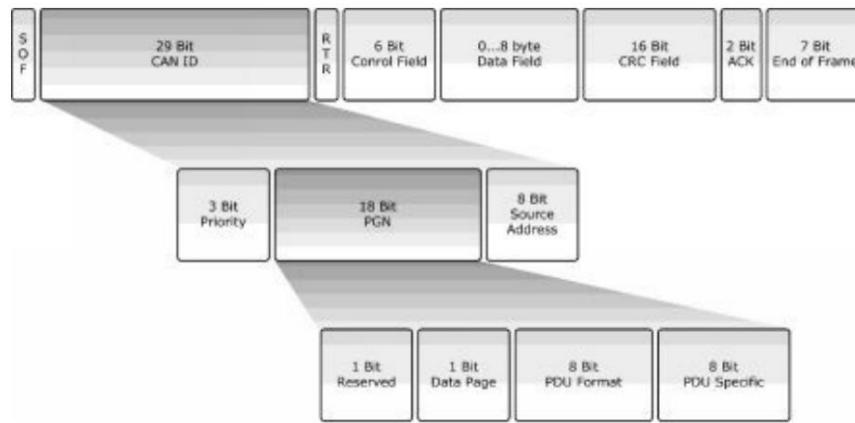
Примечание. Вам понадобится копия стандарта SAE J1939/71 для подробного описания всех PGN и SPN. Вы можете найти некоторые из них, прошагивая по Интернету, но имейте в виду, что полного онлайн-правочника нет. Описание всех PGN выходит за рамки этой книги.

2.3 Номера групп параметров (PGN)

SAE J1939 — очень оригинально разработанный протокол, ис пользующий изобретательное право на 29-битный идентификатор с сообщения CAN. Вместо этого, чтобы полагаться на множество функций протокола, SAE J1939 ис пользует предопределенные таблицы параметров, что поддерживает фактический протокол на понятном уровне.

Группами параметров являются, например, температура двигателя, которая включает температуру масла и т. д. Группы параметров и их номера (PGN) перечислены в SAE J1939 (примерно 300 страниц) и определены в SAE J1939/71, документ, содержащий примерно 800 страниц, заполненных определениями групп параметров, а также подозрительными номерами параметров (SPN). Кроме того, можно ис пользовать группы параметров, специфичные для производителя.

С точки зрения программирования важно знать, что комбинация PGN, приоритета с сообщения и адреса узла (источника) приложения предоставляет с собой 29-битный идентификатор с сообщения CAN, как показано на следующем рисунке.



Примечание. Более подробную информацию о формате с сообщения J1939 можно найти в официальной документации SAE или в моей книге «Понятие руководство по J1939». Однако определения, например, «Формат PDU» и «Специфический PDU» могут быть для новичка J1939, т. е. образовательная ценность сомнительна.

2.3.1 Группы параметров

Группы параметров содержат информацию о назначении параметров в пределах 8-ми

байтовое поле данных CAN каждого сообщения, а также частота повторения и приоритет.

Ниже приведен пример определения группы параметров, как указано в SAE.

J1939/71:

- PGN 65262 - Температура двигателя
 - Скорость передачи: 1 сек.
 - Длина данных: 8 байт
 - Приоритет по умолчанию: 6
 - Номер PG: 65262 (FEEEhex)
- Описание данных
 - Байт 1: Температура охлаждения жидкости двигателя — SPN 110
 - Байт 2: Температура топлива — SPN 174
 - Байт 3, 4: Температура моторного масла — SPN 175
 - Байт 5, 6: Температура масла турбонагнетателя — SPN 176
 - Байт 7: Температура промежуточного воздуха ладителя двигателя — SPN 52
 - Байт 8: открытие термостата промежуточного воздуха ладителя двигателя — SPN 1134

2.3.2 Подозрительные номера параметров (SPN)

Номер подозрительного параметра (SPN) — это номер, присвоенный SAE конкретному параметру в группе параметров. Он подробно описывает параметр, предоставляемый следующую информацию:

- Длина данных в байтах
- Тип данных
- Разрешение
- Компенсировать
- Диапазон
- Сылочный тег (метка)

SPN с общими характеристиками группируются в группы параметров (PG) и передаются вместе с использованием номера группы параметров (PGN).

Продолжая предыдущий пример (PGN 65262), параметр Engine

Температура ох лаждаю щей жидкости описывается SPN 110 следующим образом:

- SPN 110 — Температура ох лаждаю щей жидкости двигателя
- Температура жидкости системы охлаждения двигателя
- Длина данных : 1 байт
 - Решение: 1 градус С/бит
 - Смещение: -40 градус С
 - Диапазон данных : от -40 до 210 градусов С
 - Тип Измеренный
 - Ссылка: PGN 65262

2.3.3 Диапазон PGN

Номера параметров программы (PGN) находятся в диапазоне:

0x0000 – 0xEEFF: 239 одноранговых сообщений, определенных SAE 0xEF00 – 0xFFFF: 1

одноранговое сообщение для частного использования 0xF000 – 0xFEFF: 3840

широковещательных сообщений, определенных SAE 0xFF00 – 0xFFFF: 256

широковещательных сообщений для частного использования

Примечание. SAE J1939 допускает два типа сообщений: одноранговые (= прямая связь между узлами) и широковещательные. Широковещательные сообщения (их идентификатор сообщения включает в себя адрес узла-отправителя) распространяются на все узлы, и узлы решают, использовать его или нет. В одноранговых сообщениях используется идентификатор сообщения, который включает адрес отправителя и получателя. Адрес узлов всегда имеет длину 8 бит.

2.4 Два элемента с тека протоколов SAE J1939

Любое оборудование SAE J1939 должно поддерживать SAE J1939/1x и SAE J1939/21, иначе они бесполезны, поскольку эти стандарты описывают физический уровень шины CAN и основные функции протокола. Эта часть в достаточной степени закрыта (т. е. в демонстрационных целях) нашим экраном CAN, в то время как кабели и разъемы могут не обязательно соответствовать стандарту J1939.

Полнфункциональное программное обеспечение протокола SAE J1939 должно поддерживать один обязательный элемент, SAE J1939/81 (протокол утверждения адреса) и, при необходимости, SAE J1939/21 (передача до 1785 байтов на сообщение).

Примечание. Реализация транспортного протокола (TP), т. е. передача до 1785 байтов данных в сообщении, сильно зависит от приложения. Некоторым ЭБУ и их управляемым приложениям (CA) это понадобится, некоторым — нет.

Далее я расскажу об основах транспортного протокола SAE J1939/21 (TP) и процедуры утверждения адреса в соответствии с SAE J1939/81. Представленная информация предназначена для помощи в понимании текущих протоколов ARD1939.

Примечание. Для получения более подробной информации о требованиях SAE J1939/21 и SAE J1939/81 см. главу «Проверка концепции». Тесты, выполненные в этой главе, объясняют фактический протокол SAE J1939 более подробно, чем любой другой учебник или руководство.

2.4.1 SAE J1939/21 — Транспортный протокол (TP)

Несмотря на то, что CAN чрезвычайно эффективен в легковых автомобилях и небольших промышленных приложениях, сам по себе CAN не подходит для удовлетворения требований связи между грузовиками и автобусами, особенно потому, что связь между устройствами ограничена в 8 байтами на сообщение. Однако можно увеличить размер сообщения CAN, внедрив дополнительное программное обеспечение, т. е. так называемые протоколы более высокого уровня. J1939 является таким протоколом более высокого уровня и поддерживает до 1785 байтов на сообщение.

Чтобы поддерживать размер более 8 байт, с сообщение необходимо упаковать в последовательность с сообщений размером 8 байт. Следовательно, получатель такого многошагового пакетного сообщения должен повторно собрать данные. Такие функции определены как функции транспортного протокола (TP) и описаны в SAE J1939/21.

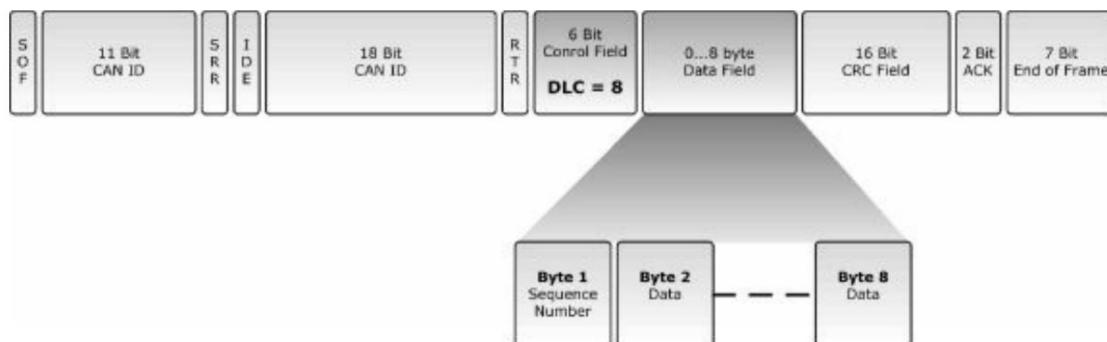
Для упаковки с сообщений CAN в последовательность до 1785 с сообщений (а также для повторной сборки кадров CAN в один пакет данных) J1939

Транспортный протокол определяется следующим:

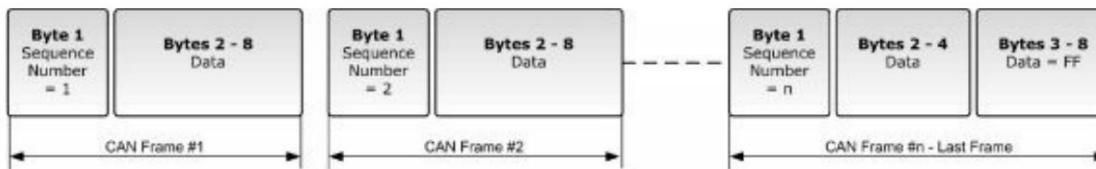
- Каждое многошаговое сообщение передается с использованием выделенного PGN для передачи данных (60160, TP.DT = передача данных по транспортному протоколу), т.е. все пакеты с сообщений будут иметь один и тот же идентификатор.
- Управление потоком осуществляется другим выделенным PGN (60146, TP.CM = управление связью по транспортному протоколу).
- Длина сообщения всегда должна быть 8 байт (DLC = 8).
- Первый байт в поле данных содержит порядковый номер в диапазоне от 1 до 255.
- Остальные 7 байт заполняются данными из одного длинного (> 8 байт) сообщения.
- Все неиспользуемые байты данных в последнем пакете устанавливаются в FFhex.

Метод использования порядкового номера плюс оставшиеся семь байтов данных дает в общей сложности (255 пакетов по 7 байт/пакет) 1785 байтов на многошаговое сообщение.

Следующие изображения демонстрируют использование поля данных CAN, включая порядковый номер и упаковку нескольких сообщений CAN.



Кадр данных CAN с порядковым номером



Пример последовательности из нескольких пакетов

Как и в случае с обычными 8-байтовыми сообщениями CAN, многопакетные сообщения (т. е. с сообщением длиннее 8 байтов) могут передаваться в виде широковещательного сообщения (BAM = широковещательное уведомление) или одноранговое сообщение (RTS/CTS = Запрос на отправку / Разрешение на отправку).

Примечание. Стандарт SAE J1939/21 требует, чтобы ECU одновременно поддерживал один сеанс BAM и один сеанс RTS/CTS.

2.4.1.1 Многопакетная широковещательная передача (с сеансом BAM)

Чтобы передать многопакетное сообщение, узел должен сначала отправить широковещательное объявление (BAM), которое содержит следующие компоненты:

- Номер группы параметров (PGN) многопакетного сообщения
- Размер многопакетного сообщения
- Количество упаковок

Сообщение BAM позволяет всем узлам-получателям (т. е. всем узлам, заинтересованным в сообщении) подготовиться к приему путем выделения соответствующего объема ресурсов (памяти).

Сообщение широковещательного объявления (BAM) встроено в транспортный протокол — управление с единением (TP.CM) PGN 60416, а фактическая передача данных обрабатывается с использованием передачи данных PGN 60160.

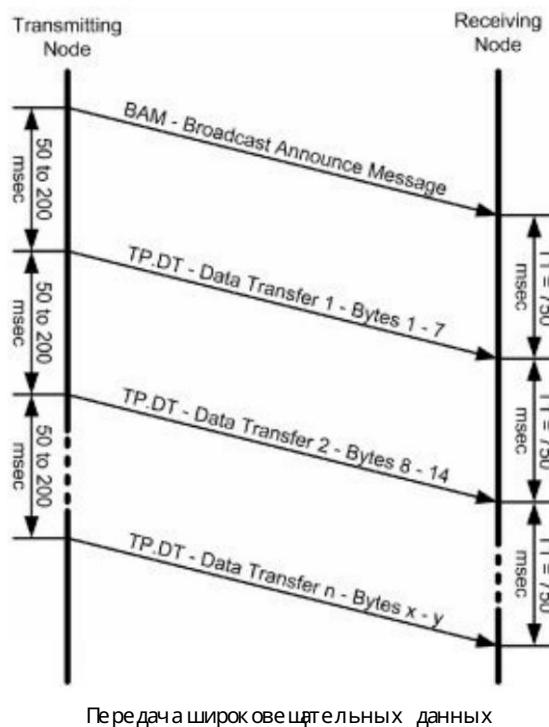
Группа параметров Имя	Транспортный протокол — управление с единением (TP.CM)
Параметр Число	Группа 60416 (0xEC00)
Определение	Используется для управления потоком управления связью (например, широковещательное объявление).

Скорость передачи	В соответствии с номером группы параметров, который необходимо передать
Длина данных	8 байт 0
Страница расширенных данных (P)	
Страница данных	0
Формат PDU	236
Конкретный PDU	Адрес назначения (= 255 для широковещательной страницы)
Приоритет по умолчанию	7
Описание данных	(Только для сообщения отображения)
Байт	1 — байт управления = 32
	2,3 — размер сообщения (количество байтов; с начала LSB, затем MSB)
	4 — общее количество пакетов
	5 — зарезервировано (должно быть заполнено 0xFF)
	6-8 — Номер группы параметров мультиплексированного сообщения (6=младший бит, 8=старший бит)
Группа параметров	
Имя	Транспортный протокол — передача данных (TP.DT)
Параметр	
Группа	60160 (0xEB00)
Число	
Определение	Передача данных многопакетных сообщений
Скорость передачи	В соответствии с номером группы параметров, который необходимо передать
Длина данных	8 байт 0
Страница расширенных данных (P)	
Страница данных	0
Формат PDU	235
Конкретный PDU	Адрес назначения
Приоритет по умолчанию	7
Описание данных	
Байт	1 — Порядковый номер (от 1 до 255)

2-8 - Данные

Для последнего пакета многопакетной PGN может потребоваться менее восьми байтов данных . Всё неиспользуемые байты данных в последнем пакете устанавливаются на 0xFF.

Передача многопакетных широковещательных с сообщений не регулируется какими-либо функциями управления потоком, поэтому необходимо определить временные требования между отправкой широковещательного сообщения (BAM) и передачей данных PGN. На следующем рисунке показана последовательность с сообщений и требования к времени для широковещательного мультипакетного сообщения .



Как показано на предыдущем изображении:

- Час тога пакетов с сообщений будет состоять от 50 до 200 мс .
- Тайм-аут возникает, когда между двумя пакетами с сообщений прошло время более 750 мс (T_1), когда ожидалось больше пакетов.

Тайм-ауты вызовут закрытие соединения .

Соединение считается закрытым, когда

- Отправитель с сообщения данных отправляет последний пакет передачи данных .
- Происходит тайм-аут.

Примечание. Документ SAE J1939/21 не очень конкретен в отношении диапазона времени от 50 до 200 миллисекунд, т.е. того, как был определен диапазон и какое время следует применять при каких условиях .

Прежде всего, минимум 50 миллисекунд создает достаточно времени для прохождения другого с сообщения по сети. Использование с сообщения с самым низким приоритетом гарантирует, что с сообщения с более высоким приоритетом будут по-прежнему с временно достигать шины. Другими словами, с учетом большого времени и низкого приоритета в определенной степени гарантирует, что длинные сообщения данных не нарушают обычный поток данных .

Однако диапазон от 50 до 200 миллисекунд до их пористости загадкой. Он может быть основан на технологии, когда был введен SAE J1939, т.е. могли быть ЭБУ с временем реакции до 200 миллисекунд. Другим, более правдоподобным объяснением может быть то, что время может быть выбрано в соответствии с потребностями приложения , это означает, что 200 миллисекунд менее важны для обычного трафика данных , чем 50 миллисекунд.

2.4.1.2 Многопакетная одноранговая связь (стандарт RTS/CTS)

Передача мультипакетных сообщений, специфичных для пункта назначения (одноранговых), подлежит управлению потоком.

1. Инициализация соединения . Отправитель с сообщения передает с сообщение «Запрос на отправку» . Принимающий узел отвечает либо с сообщением Clear to Send , либо с сообщением Connection Abort , если он решает не устанавливать соединение. Прерывание соединения в качестве ответа на сообщение «Запрос на отправку» предпочтительнее тайм-аута для инициатора соединения . С сообщение «Готов к отправке» содержит количество пакетов, которые ожидает получатель, плюс ожидаемый порядковый номер.
2. Передача данных . Отправитель передает PGN передачи данных после получения с сообщения «Готов к отправке» . Передача данных может быть прервана/остановлена с сообщением Connection Abort .
3. Закрытие соединения . Получатель с сообщения после получения последнего пакета с сообщения отправляет подтверждение завершения с сообщения .

(подтверждение) при условии, что во время передачи не было ошибок. Любой узел, отправитель или получатель, может отправить сообщение о разрыве соединения. Причиной прерывания соединения может быть тайм-аут.

Надежное управление потоком также должно включать тайм-ауты для обеспечения правильной работы сети, и SAE J1939/21 определяет количество тайм-аутов.

$T_r = 200$ мс — время отклика

$T_h = 500$ мс — время задержки

$T_1 = 750$ мс

$T_2 = 1250$ мс

$T_3 = 1250$ мс

$T_4 = 1050$ мс

Сценарии для управления тайм-аутом:

- Узел (независимо от того, является ли узел получателем или отправителем сообщения данных) не отвечает в течение 200 мс (T_r) на сообщение данных или управление потоком.
- Если принимающему узлу необходимо (по какой-либо причине) задержать передачу данных, он может отправить сообщение «Готов к отправке», в котором количество пакетов установлено равным нулю. В случае, когда поток должен быть задержан на определенное время, получатель с сообщения должен повторять передачу с сообщения «Готов к отправке» каждые 0,5 секунды (T_h), чтобы поддерживать открытое соединение с отправителем с сообщения. Как только получатель будет готов принять сообщение, он должен отправить обычное сообщение «Готов к отправке».
- Между двумя пакетами с сообщений прошло время, превышающее T_1 , хотя ожидалось больше пакетов.
- Время, превышающее T_2 , истекло после сообщения «Готов к отправке» без получения данных от отправителя данных.
- Время, превышающее T_3 , истекло после того, как последний переданный пакет данных не получил с сообщения «Готов к отправке» или «Подтверждение окончания с сообщения» (ACK).
- Время, превышающее T_4 , истекло после отправки сообщения «Готов к отправке», чтобы отложить передачу данных без отправки другого сообщения «Готов к отправке».

Следовательно, любое условие таймаута приведет к закрытию соединения.

Примечание. Что не упоминается в SAE J1939/21, так это то, что сообщение «Готов к отправке» может быть отправлено получателем сообщения с данными в любое время либо сразу после получения сообщения «Запрос на отправку», либо после получения пакета данных, то есть любое время во время передачи данных.

Другими причинами закрытия соединения являются:

- Отправитель с сообщениями данных отправляет последний пакет передачи данных.
- Получатель с сообщениями данных получает последний пакет передачи данных, и происходит одит таймаут T1.
- Отправитель с сообщениями данных получает сообщение ACK End of Message.
- Получение сообщения о разрыве соединения.

Сообщения управления потоком, такие как Request to Send, Clear to Send и т. д., встроены в транс портный протокол — управление с соединением (TP.CM) PGN 60416, а фактическая передача данных обрабатывается с использованием Data Transfer PGN 60160.

Имя группы параметров Транс портный протокол — управление с соединением (TPCM)	
Номер группы параметров 60416 (0x00EC00)	
Определение	Используется для управления потоком управления связью (например, запрос на отправку, разрешение на отправку и т. д.).
Скорость передачи	В соответствии с номером группы параметров, который необходимо передать
Длина данных	8 байт 0 0
Страница расширенных данных (R)	
Страница данных	
Формат PDU	236
Конкретный PDU	Адрес назначения (= 255 для широковещательной страницы)
Приоритет по умолчанию	7
Описание данных	В зависимости от содержимого контрольного байта — см. следующее описание.

TP.CM_RTS	<p>Запрос режима с оединения на отправку 1 —</p> <p>управляю щий байт = 16 2,3</p> <p>— размер с сообщения (количество байтов)</p> <p>4 – Общее количество пакетов 5 –</p> <p>Макс . количество пакетов в ответ на CTS.</p> <p>Без ограничений при заполнении 0xFF.</p>
	6-8 – Номер группы параметров мультипакетного сообщения (6=LSB, 8=MSB)
TP.CM_CTS	<p>Режим подключения Готов к отправке</p> <p>1 — байт управления = 17</p> <p>2 - Общее количество пакетов (не должно превышать байт 5 в RTS)</p>
	3 – номер следующего пакета
	4,5 — зарезервировано (должно быть заполнено 0xFF)
	6-8 – Номер группы параметров мультипакетного сообщения (6=младший бит, 8=старший бит)
TP.CM_EndOfMsgACK Подтверждение окончания сообщения	<p>1 — управляю щий байт</p> <p>= 19 2,3 — размер с сообщения</p> <p>(количество байтов)</p> <p>4 – Общее количество пакетов 5 –</p> <p>Зарезервировано (должно быть заполнено 0xFF)</p>
	6-8 – Номер группы параметров мультипакетного сообщения (6=LSB, 8=MSB)
TP.Conn_Abort	<p>Прерывание с оединения</p> <p>1 — байт управления = 255</p> <p>2 – Причина разрыва с оединения (см. следующее описание)</p>
	3-5 — зарезервировано (должно быть заполнено 0xFF)

6-8 – Номер группы параметров мультиплексированного сообщения

(6=младший бит, 8=старший бит)

Байт управления = 32 зарезервирован для широковещательного сообщения. Байты управления 0-15, 18, 20-31, 33-254 зарезервированы SAE.

Причинами прерывания соединения могут быть:

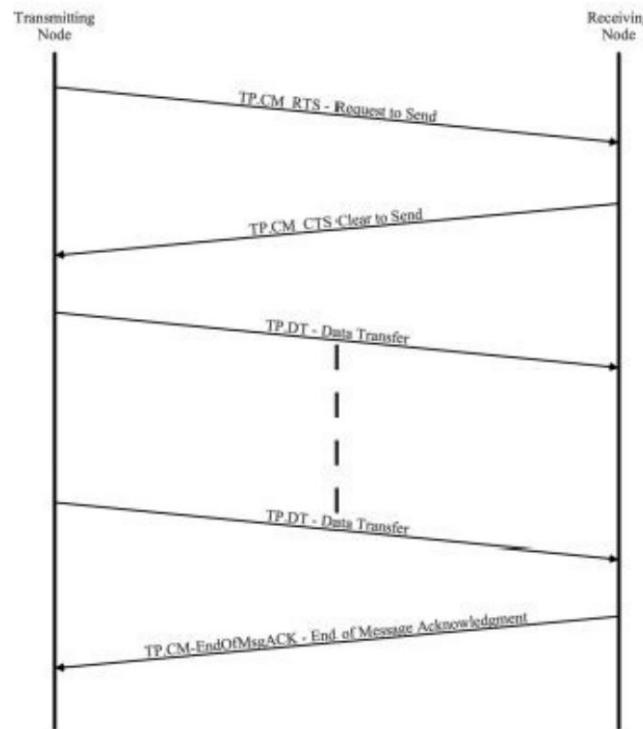
- 1 — узел уже занят другим сервисом и не может поддерживать другое соединение.
- 2 — Узлу не хватает необходимых ресурсов.
- 3 — тайм-аут.
- 4..250 — зарезервировано SAE.
- 251..255 — в соответствии с определениями J1939/71 (с ожиданием, с борником стандартов SAE J1939 не содержит дополнительных полей).

Фактическая передача данных осуществляется с помощью Data Transfer PGN 60160.

Группа параметров	Транспортный протокол — передача данных (TP.DT)
Имя	
Параметр	Группа
Число	60160 (0x00EB00)
Определение	Передача данных многопакетных сообщений
Скорость передачи	В соответствии с номером группы параметров для передачи 8 байт
Длина данных	
Страница расширенных данных (P)	0
Страница данных	0
Формат PDU	235
Конкретный PDU	Адрес назначения
Приоритет по умолчанию	7
Описание данных	
Байт	1 — Порядковый номер (от 1 до 255)
	2-8 — Данные

Для последнего пакета многопакетной PGN может потребоваться я меньше восемьми байтов данных. Всё неиспользуемые байты данных в последнем пакете устанавливаются на 0xFF.

На следующем изображении показан новый поток данных между режимами передачи и приема во время сеанса RTS/CTS (т. е. одноранговой связи).



Одноранговая передача данных

Примечание. Я воздержался от вставки различных тайм-аутов, которые могут возникнуть во время сеанса RTS/CTS. Для каждого отдельного случая тайм-аута потребуется свой блок-схема, а функциональность уже достаточно описана.

2.4.2 SAE J1939/81 — Процедура подачи претензии по адресу

В то время как другие протоколы более высокого уровня, основанные на сети контроллеров (CAN), не поддерживают назначение адресов узлов по умолчанию, протокол SAE J1939 предоставляет еще одну оригинально разработанную функцию для уникальной идентификации блоков управления электроникой (ECU) и их основной функции.

Примечание. Напоминаем, что сетевой узел J1939 обычно называется «ECU». Каждый

ECU может поддерживать более одного «приложения управления (СА)». По этой причине технические требования обычно относятся к СА, а не к ECU.

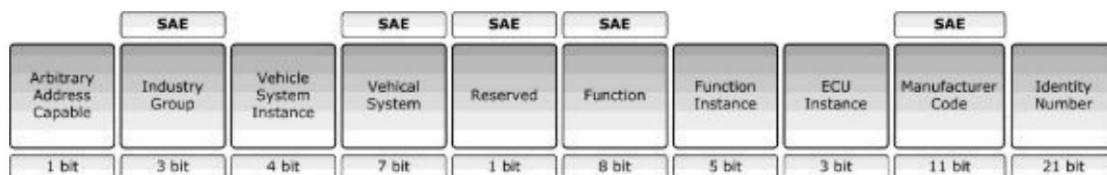
2.4.2.1 Технические требования

Стандарт SAE J1939/81 перечисляет ряд технических требований к процессу запроса адреса:

1. Каждое управляющее приложение (СА) должно обеспечивать уникальное 64-битное ИМЯ.
2. Центры сертификации должны успешно звать адрес перед отправкой сообщений (кроме сообщений для запроса адреса) в сеть.
3. Несмотря на то что запросить адрес должна быть обработана и сообщена в сеть.
4. Центр сертификации должен выполнить инициализацию связанную с процессом подачи заявки на адрес.
5. ЦС должен поддерживать минимальный набор требований к управлению сетью, включая необходиимые реакции на перебои в подаче электроэнергии.

2.4.2.2 Имя устройства

Стандарт J1939 определяет 64-битное ИМЯ, как показано на следующем рисунке, для уникальной идентификации ECU в сети.



Как показано на изображении выше, некоторые параметры назначаются SAE, остальные устанавливаются производителем/разработчиком приложения.

- Возможность произвольного адреса — указывает, может ли ECU/СА сгенерировать адрес (1 = да; 0 = нет). Некоторые ECU могут поддерживать только один адрес; другие поддерживают диапазон адресов.
- Отраслевая группа — эти коды связанны с конкретными отраслями, такими как дорожное оборудование, сельское хозяйство и телекоммуникации,

и более.

- Экземпляры системы транс портног ос редс тва — присваивает номер каждому экземпляру в системе транс портног ос редс тва (в случае подключения нескольких сетей — например, с единения вагонов в поезде).
- Система транс портног ос редс тва. Системы транс портног ос редс тва связанны с отраслевой группой могут быть, например, «тягач» в «Общероссийской» отрасли или «прицеп» в отраслевой группе «Транспорт».
- Зарезервировано — все равно.
- Функция — этот код в диапазоне от 128 до 255 назначается в соответствии с отраслевой группой. Значение от 0 до 127 не связано никаким другим параметром.
- Экземпляр ECU — есть J1939 может содержать несколько ECU одного типа (т. е. с одинаковыми функциями). Код экземпляра разделяется.
- Код производителя — 11-битный код производителя присваивается SAE.
- Идентификационный номер — это поле присваивается производителем аналогично серийному номеру, т. е. код должен быть уникальным для устройства.

В целях тестирования и моделирования, а также при условии, что NAME используется только для процесса с аутверждения адреса, можно установить все необходимые параметры по своему вкусу. Однако, если вы подключаете свое приложение к реальной сети SAE J1939, настоятельно рекомендуется следовать официальным стандартам, включая получение кода производителя.

Для наших проектов Arduino я назначил поля NAME таким образом, чтобы они не мешали при использовании существующей автомобильной сети. Это было сделано путем установки максимального значения идентификационного номера кода производителя, что приведет к более пастивной роли в процессе запроса адреса ECU с более высоким значением NAME с большей вероятностью проигрывает конкуренту другому узлу, используя ту же адрес.

Примечание. Всё показанные настройки используются только в демонстрационных целях. В результате вы должны следовать рекомендациям SAE. Кроме того, только вы (а не автор или издатель) несете ответственность за окончательную реализацию и ее результаты.

```
#define NAME_IDENTITY_NUMBER 0xFFFFFFF #define NAME_MANUFACTURER_CODE  
0xFFF #define NAME_FUNCTION_INSTANCE 0 #define NAME_ECU_INSTANCE  
0x00 #define NAME_FUNCTION 0xFF #define NAME_RESERVED 0 #define  
NAME_VEHICLE_SYSTEM 0x7F #define NAME_VEHICLE_SYSTEM_INSTANCE 0  
#define NAME_INDUSTRY_GROUP 0x00 #define  
NAME_ARBITRARY_ADDRESS_CAPABLE 0x01
```

Примечание. Эти настройки можно найти в файле ARD1939.h каждого проекта, в котором задействована библиотека ARD1939.

2.4.2.3 Предпочтительный адрес

В целях быстрого процееса запроса адреса каждое управляющее приложение должно поддерживать предпочтительный адрес. SAE J1939/81 рекомендует перепрограммировать предпочтительный адрес (т. е. адрес, который ECU/CA пытается получить при включении питания), чтобы обеспечить правильную настройку всей сети автомобиля. Это также помогает предотвратить задержки/проблемы во время процеедуры запроса адреса. Надлежащий процедурой было бы определить все предпочтительные адреса во всей сети и смотреть на них так, чтобы не возникло коллизий.

Согласно стандарту SAE J1939/81, каждый центр сертификации должен пытаться использовать предпочтительные адреса, назначенные SAE в соответствии с отраслевой группой, что означает, что каждая отраслевая группа имеет список предпочтительных адресов в соответствии с функцией центра сертификации.

В случае глобальной отраслевой группы эти адреса находятся в диапазоне от 0 до 84, а диапазон от 128 до 247 зависит от отраслевой группы. Согласно

Стандарт SAE J1939/81, диапазон от 85 до 127 зарезервирован для будущего присвоения SAE.

Кроме того, в соответствии с SAE J1939/81 управляющее приложение (CA), требующее предпочтительного адреса в диапазоне от 0 до 127 и от 248 до 253, должно выполнять функцию, определенную для этого процесса, и указывать эту функцию в своем имени.

Все эти определения могут сбиваться с толку и даже противоречить инженеру, когда дело доходит до тестирования ЭБУ J1939. В демонстрационных целях (например, в наших проектах Arduino) мы предположим предпочтительный адрес 128 (0x80) и диапазон адресов между 129 и 247.

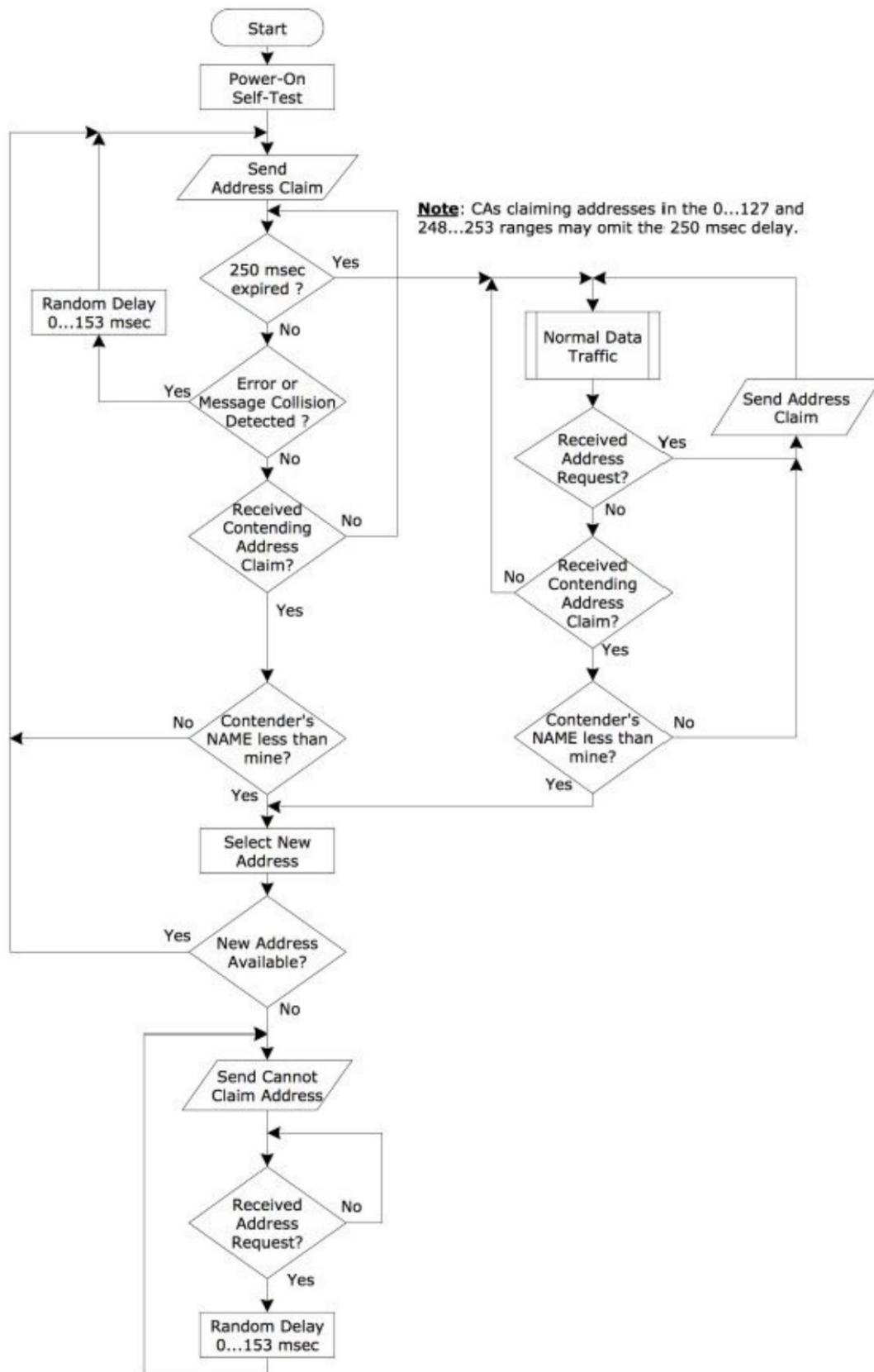
2.4.2.4 Процедура утверждения адреса

В дальнейшем мы сосредоточимся только на двух основных сообщениях : Request Message и Address Claimed (точка же, более подробную информацию см. в рекомендациях по литературе в приложении).

Сообщение запроса использует ЦС для запроса информации, такой как ИМЯ и адреса, от всех других ЦС во всей сети. После получения сообщения запроса заявленного адреса каждый ЦС передает сообщение заявленного адреса, содержащее запрошеннную информацию. Этую процедуру можно использовать для сбора информации из сети либо для диагностики/отображения узлов, либо даже для определения подходит ли адрес узла.

Другой способ установить адрес узла — послать в шину сообщение Address Claimed . Это эквивалентно опросу о том, занят ли предпочтительный адрес . Если ответ не получен, можно с уверенностью предположить, что адрес доступен. Если есть ответ, CA должен изменить адрес и отправить другое сообщение Address Claimed .

Следующий блок-схема демонстрирует процедуру утверждения адреса:



Примечание. Процедура получения адреса, показанная на изображении, полностью реализована.

в библиотеке с текста протоколов ARD1939.

2.5 Критерии соответствия стандарту SAE J1939

Один вопрос, который неоднократно поднимался, касается соответствия SAE J1939. К сожалению, в сорнике стандартов SAE J1939 не указано, в какой момент устройство действительно может считаться соответствующим требованиям, а также нет официального тестового документа или испытательного центра.

На рынке появляется все больше устройств и программного обеспечения SAE J1939, но большинство из них не раскрывают свой уровень соответствия стандарту SAE J1939. Тем не менее, другие пытаются со списком поддерживаемых стандартов J1939, пытаясь отделить их от конкурентов.

По этой причине давайте взглянем на официальные документы SAE: Упомянутые стандарты (например, SAE J1939/21, SAE J1939/81 и т. д.) включают в себя часть Справления по стандартам J1939. «Руководство по стандартам SAE Truck and Bus Control & Communications Network — издание 2007 г.» предсказывает собой каталог с алфавитным трудом объемом около 1600 страниц, из которых около 1000 страниц относятся к таким данным, как назначения групп параметров, назначение адресов и идентификаторов, номера групп параметров и т. д.

Он был разработан в соответствии с 7-уровневой эталонной моделью ISO/OSI, где каждый уровень расматривается в соответствии со стандартом документе, как показано на следующем изображении.



SAE назвал документы, относящиеся к транс портному (4), сеансовому (5) и презентационному уровню (6) в 7-уровневой эталонной модели ISO/OSI. Эти стандарты, однако, не задокументированы (как следствие, они не нужны для J1939) и поэтому соответствующие документы не созданы.

Стандарты SAE J1939/1x и SAE J1939/2x описывают физический уровень (проводка) и канальный уровень (функции контроллера CAN). Кроме того, стандарт SAE J1939/21 описывает транс портный протокол (TP), отвечающий за транс портировку сообщений, содержащих более восьми байтов данных. Стандарт SAE J1939/81 описывает процесс утверждения адреса. По правде говоря (и я упоминал об этом ранее), это все, что нужно приложению SAE J1939.

Другими словами, любое оборудование SAE J1939 должно поддерживать SAE J1939/1x и SAE J1939/21, иначе они бесполезны, поскольку эти стандарты описывают физический уровень шины CAN и новые функции протокола. Любое программное или микропрограммное обеспечение SAE J1939 должно поддерживать SAE J1939/81 (процесс утверждения адреса) и, при необходимости, SAE J1939/21 (передача до 1785 байтов на сообщение).

Примечание. Реализация транс портного протокола (TP), т. е. передача до 1785 байтов данных в сообщении, сильно зависит от приложения. Некоторым ЭБУ и их управляемым приложениям (CA) это понадобится, некоторым — нет.

Все, что вышеходит за рамки SAE J1939/21 и SAE J1939/81 (например, диагностика SAE J1939/73, относящаяся к прикладному уровню), является краем торта, но не является частью фактического протокола.

Некоторые поставщики бизнеса J1939 заявляют о «соответствии» стандарту SAE J1939/71, хотя это является чистой рекламой. SAE J1939/71 описывает номера групп параметров (PGN), но хотя PGN передаются в соответствии с J1939, они, за некоторыми исключениями, не являются функциями протокола. PGN для управления протоколами, такими как транс портный протокол (TP) и утверждение адреса, определены в SAE J1939/21 и SAE J1939/81.

Также рекомендуется проявлять осторожность в отношении требований о соответствии SAE J1939/81 (процедура обращения с адресом). Многие устройства на рынке будут работать только с одним же статическим идентификатором узла (который по-прежнему соответствует стандарту). Однако проблема начинается, когда этот единственный адрес конфликтует с существующей сетью транс портных средств, где этот конкретный адрес уже занят (я работал с

устроиства, которым было все равно, и они продолжали отправлять данные, даже если их адрес конфликтовал с адресом другого устройства).

Примечание. Стек протоколов ARD1939, предстаивленный в одной из последующих глав, полностью поддерживает стандарт SAE J1939/81. Он может сформировать один адрес или диапазон адресов и удалить себя из шины в случае конфликта адресов. Он также поддерживает полный транспортный протокол (TP) в соответствии с SAE J1939/21.

3. Приложения SAE J1939 с Arduino

В общем, есть три разных намерения для подключения к сети транспортных средств J1939:

1. Просмотр мониторинга, обработка и отображение сетевого трафика данных.
2. Всё функции, как описано в пункте 1., но расширены возможностью отправки данных на шину J1939.
3. Всё функции, как описано в пунктах 1 и 2, но расширены транспортным протоколом J1939, поддерживающим сообщения с более чем 8 байтами данных.

Давайте рассмотрим эти три подхода:

1. Просмотр мониторинга, обработка и отображение трафика сетевых данных не требует каких-либо функций управления сетью (т. е. фактического протокола SAE J1939). Просмотрите подключите свое устройство мониторинга к шине автомобиля и отфильтруйте те сообщения, которые соответствуют вашим потребностям.
2. Для возможности отправки или запроса данных на шину J1939 или с ней требуется, чтобы ваша система (ЭБУ = электронный блок управления) владела 8-битным адресом. Поскольку адрес в автомобильной сети J1939 является динамическим (т. е. они могут меняться после каждого отключения питания), ваша система должна следовать процедуре утверждения адреса J1939 в соответствии с SAE J1939/81.
3. Транспортный протокол J1939 (SAE J1939/21) — это часть функциональности управления сетью, которая повышает уровень безопасности. Однако подавляющее большинство приложений J1939 не требуют транспортного протокола (TP). TP в основном используется для некритичных коммуникационных задач с интенсивным использованием данных.

Следующая глава посвящена проектам мониторинга и моделирования J1939, которые не требуют реализации протокола J1939. В следующей главе будет представлен стандартный протокол ARD1939, обеспечивающий полную связь с автомобильной сетью J1939.

3.1 Мониторинг и моделирование SAE J1939

Как я уже писал ранее, обе версии Arduino, используемые в этой книге, Uno и Mega 2560, хорошо подходят для проектов мониторинга и моделирования SAE J1939. До сих пор моделирование трафика данных J1939 требовало покупки дорогих устройств моделирования и связанных с этим крутых кривых обучения. Всё это меняется благодаря простым в использовании аппаратным и программным решениям Arduino.

Написание примеров мониторинга и моделирования J1939 для Arduino занимает не сколько часов или даже меньше, если в качестве шаблона использовать следующие примеры программирования.

Примечание. Всё проекты Arduino (эскизы), представленные ниже, будут использовать последовательный монитор с частотой передачи данных 115 200 бит/с (см. также мои комментарии в главе «Последовательный интерфейс и интерфейс CAN»).

3.1.1 Пример моделирования SAE J1939

Следующий пример программирования эффективно демонстрирует, насколько просто может быть моделирование трафика данных SAE J1939 (в то время как уровень сложности можно легко настроить в соответствии с требованиями приложения).

Однако, прежде чем мы погружимся в реальное кодирование, нам нужно спроектировать с моделированные сообщения, т. е. нам нужно определить 29-битный идентификатор сообщения CAN, который включает фактический PGN, приоритет PGN и его однозначный адрес приложения.

3.1.1.1 Дизайн с сообщениями

Для этого примера программирования я с лучшим образом выбрал два PGN из огромного набора доступных номеров групп параметров. Они есть:

- PGN 65267 — Положение автомобиля, указывает широту и долготу автомобиля.
 - Частота повторения передачи: 5 с Номер группы параметров: 0xFFE3 Длина данных: 8
 - Байт Приоритет по умолчанию: 6

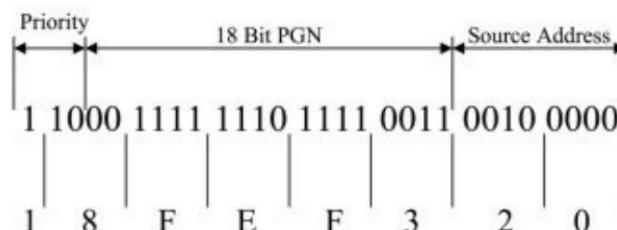
- PGN 65269 - Условия окружающей среды, обес печивает атмосферное давление, температуру внутри кабины, температуру окружающей воздуха, температуру воздуха на входе в двигатель и температуру поверхности дороги.
 - Частота повторения передачи: 1 с Номер группы параметров: 0xEF5 Длина данных : 8 байт
 - Приоритет по умолчанию : 6
 -

Примечание. Для реализации других или дополнительных PGN вам потребуется копия стандартного SAE J1939/71 для подробного описания параметров и формата данных. Вы можете найти некоторые из них, просматривая Интернет, но имейте в виду, что полного онлайн-правочника нет. Описание всех PGN выходит за рамки этой книги.

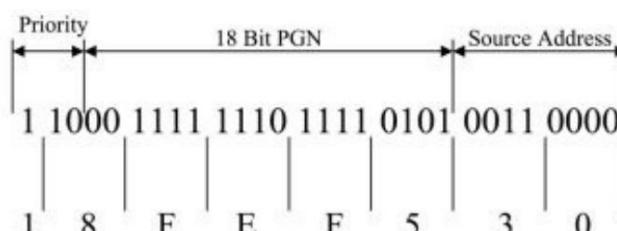
Помимо фактических PGN, нам также потребуется с моделировать их одинаковый адрес приложения. Их одинаковый адрес обычно определяется в процессе утверждения адреса стандартных протоколов SAE J1939, который еще не является частью этого примера программирования. Для простых тестовых демонстрационных целей достаточно предположить адреса. Далее мы используем 0x20 для PGN 65267 и 0x30 для PGN 65269.

Применяя PGN плюс приоритет плюс адрес источника, идентификаторы сообщений будут следующими:

- PGN 65267 — Позиция автомобиля : 0x18FEF320



- PGN 65269 — Внешние условия : 0x18FEF530



3.1.1.2 Код Arduino

Код Arduino довольно прост и практически не требует пояснений. Однако позвольте мне упомянуть несколько слов о структуре программы.

Прежде всего, в зависимости от щита CAN, используемого для этого примера программирования, убедитесь, что вы правильно установили CS (выбор чипа) для контроллера CAN, как описано в модуле can.cpp (см. также мои замечания в главе о CAN-контроллере). BUS Shield от Seeed Studio).

В основном программном модуле J1939_Data_Traffic_simulation я определил два сообщения, msgVehiclePosition и msgAmbientConditions, и заполнил их с лучшими данными.

В основном цикле я использую функцию задержки для сознания одновременного таймера с помощью переменной nCounter. Содержимое nCounter запускает передачу соответствующих сообщений.

Фактический программный код довольно прост:

```
недействительный цикл
{
    // Установить базовую задержку
    таймера delay(1000); // 1 сек
    nCounter++;

    // Отправляем PGN 65629 — внешние условия
    canTransmit(0x18FEF530, msgAmbientConditions, 8);

    // Отправляем PGN 65627 - Положение автомобиля if(nCounter
    == 5)

    { canTransmit(0x18FEF320, msgVehiclePosition, 8); nСчетчик = 0; // Сброс
    счетчика

    } // конец , если

} // конец цикла
```

Примечание. Этот проект Arduino доступен на странице загрузки по адресу <http://ard1939.com>.

3.1.1.3 Подтверждение концепции

Результат был протестирован с помощью шлюза ADFweb CAN-to-USB в сочетании с его инструментом анализа CAN на базе Windows. Как показано на скриншоте экрана ниже, PGN 65267 (положение автомобиля : 0x18FEF320) принимается каждые 5 секунд, а PGN 65269 (окружающие условия : 0x18FEF530) появляется каждую секунду (в соответствии с отметками времени).

NR	TIME	ID (HEX)	DATA (HEX)	ASCII
1	12:15:48 AM.268.1	18FEF530	38 37 36 35 34 33 32 31	87654321
2	12:15:49 AM.266.5	18FEF530	38 37 36 35 34 33 32 31	87654321
3	12:15:50 AM.264.9	18FEF530	38 37 36 35 34 33 32 31	87654321
4	12:15:51 AM.263.2	18FEF530	38 37 36 35 34 33 32 31	87654321
5	12:15:52 AM.261.6	18FEF530	38 37 36 35 34 33 32 31	87654321
6	12:15:52 AM.262.1	18FEF320	31 32 33 34 35 36 37 38	12345678
7	12:15:53 AM.260.2	18FEF530	38 37 36 35 34 33 32 31	87654321
8	12:15:54 AM.258.5	18FEF530	38 37 36 35 34 33 32 31	87654321
9	12:15:55 AM.256.9	18FEF530	38 37 36 35 34 33 32 31	87654321
10	12:15:56 AM.255.3	18FEF530	38 37 36 35 34 33 32 31	87654321
11	12:15:57 AM.253.6	18FEF530	38 37 36 35 34 33 32 31	87654321
12	12:15:57 AM.254.2	18FEF320	31 32 33 34 35 36 37 38	12345678
13	12:15:58 AM.252.2	18FEF530	38 37 36 35 34 33 32 31	87654321
14	12:15:59 AM.250.6	18FEF530	38 37 36 35 34 33 32 31	87654321

3.1.1.4 Расширенная версия программы

Хотя предыдущий пример программирования доказывает способность разработки симулятора трафика данных SAE J1939, я решил не обходимо улучшить код таким образом, чтобы упростить расширение и модификацию моделирования. Две эти элементами, которые нуждались в улучшении, были передача сообщения (которая требует состояния идентификатора сообщения из PGN, приоритета и адреса источника) и управлениетаймером (довольно элементарное).

В этой расширенной версии я заменил функцию canTransmit на j1939Transmit, которая позволяет передавать PGN, приоритет и исходный адрес как отдельные параметры.

Итак, вместе:

- canTransmit (0x18FEF530, msgAmbientConditions,
- 8); canTransmit(0x18FEF320, msgVehiclePosition, 8);

- j1939Transmit(65269, 6, 0x30, msgAmbientConditions, 8);
- j1939Transmit(65267, 6, 0x20, msgVehiclePosition, 8);

Управление таймером добавляет более высокий уровень сложности, но улучшает адаптивность кода. Прежде всего, нам нужно объявить ис пользованные таймеры:

- с помощью j1939Timer pTimerVehiclePosition;

В процессе нас тройки таймеры не обходимо инициализировать:

- j1939TimerReset(&pTimerVehiclePosition);
pTimerVehiclePosition.nCount = 5000;
pTimerVehiclePosition.bStart = ис тина;

В функции цикла мы должны вызвать функцию управления таймером, которая ис пользует базу времени в 1 миллисекунду:

- j1939TimerControl();

Также в функции цикла мы должны проверить таймер:

- если (pTimerVehiclePosition.bExpired == true) {

j1939TimerReset(&pTimerVehiclePosition); // Сброс таймера
j1939Transmit(65267, 6, 0x20, msgVehiclePosition, 8); // Передаем
pTimerVehiclePosition.nCount = 5000; // Переапустить таймер
pTimerVehiclePosition.bStart = true; } //
конец , если

Модифицированная версия таймера обеспечивает более точное разрешение таймера, что также улучшает любое время реакции приложения, например, когда речь идет о приеме кадров с сообщений SAE J1939 и изменениях передаваемых данных в соответствии с полученной информацией.

В этом случае я воздержусь от показа другого снимка экрана в качестве доказательства концепции. Изображение идентично изображению в предыдущей главе.

Примечание. Этот проект Arduino доступен на странице загрузки по адресу

<http://ard1939.com>.

3.1.1.5 Симулятор напряжения SAE J1939

Повышенная гибкость в сочетании с более высоким разрешением таймера, которого мы достигли в предыдущем проекте, позволяет нам легче реализовать другую задачу, а именно симулятор с тресом.

Симулятор с тресом в данном случае означает увеличение нагрузки на шину трансポートного срода и, таким образом, «нагрузочное тестирование» определенного устройства J1939. Многие разработчики задаются вопросом, сможет ли их устройство справиться с повышенной нагрузкой на шину и при этом функционировать в соответствии со спецификацией.

В качестве тестового предупреждения, следующий проект имеет с вою ограничения с точки зрения достижения максимальной загрузки шины (100%) из-за аппаратных ограничений (время часов Arduino) и максимального разрешения таймера в 1 миллисекунду. Время передачи сообщения J1939 составляет примерно 540 микросекунд, а отправка его каждую миллисекунду приведет к максимальной загрузке шины в 50%.

Отправка двух последовательных сообщений в течение одного миллисекундного цикла может легко увеличить нагрузку на шину, но это также тот случай, когда мы достигаем аппаратных ограничений. Дальнейшие улучшения могут быть достигнуты за счет реализации подпрограммы обслуживания прерывания для передачи сообщений и более точного разрешения таймера, чем то, которое обеспечивается функцией `delay()`. Это, однако, привело бы к значительной корректировке кода, и если не стоили бы результата.

Самое простое решение для максимальной нагрузки на шину — запустить одну и ту же программу на двух платах Arduino, подключенных к одной и той жешине.

Примечание. Загрузка шины SAE J1939 в 60% считается чрезвычайно высокой (100% будет непрерывным потоком данных) и, таким образом, является достаточной для стресс-теста.

В проекте J1939_Stress_Test я просто скопировал предыдущий проект и использовал ту же PGN для передачи. Однако в функции `loop()` я уменьшил значения таймера до 1 миллисекунды, что означает, что программа будет отправлять два сообщения в течение одной миллисекунды. Согласно моему программному обеспечению для анализа CAN ADFweb, это создает нагрузку на шину 57%.

В другом варианте я загрузил эту же программу на другой одновременно подключенную к той же сети Arduino, однако поигрался с количеством передаваемых PGN и их частотой:

- 1 сообщение каждые 2 миллисекунды: загрузка шины = 76%
- 1 сообщение каждую 1 миллисекунду: загрузка шины =
- 79% 2 сообщения каждые 1 миллисекунда: загрузка шины = 79%

Я предполагаю, что предел интерфейса ADFweb составляет 79%, чтобы вляется неплохим значением. Из-за внутренних времен задержки вы вряд ли найдете какой-либо шлюз CAN, который приближается к 100%, если только вы не готовы потратить большие деньги.

Я воздержусь от размещения здесь кода проекта, так как он почти идентичен предыдущему.

Примечание. Этот проект Arduino доступен на странице загрузки по адресу <http://ard1939.com>.

3.1.2 Примеры мониторинга SAE J1939

Следующая глава в первую очередь посвящена мониторингу данных J1939, т. е. описанные проекты Arduino работают в основном в режиме «только прослушивание». Любой двусторонний связь сшиной трансポートного средства требует реализации соответствующих протоколов J1939, в частности, процесса аутверждения адреса. Без адреса узла (источника) доступ к шине автомобиля может привести к конфликту адресов и неисправностям.

Следующие проекты были разработаны с предположением, что использование предполагаемого адреса узла не повлияет на есть ли из-за лабораторных условий, ни из-за знания адресов всех узлов в сети трансポートного средства (таким образом предотвращая конфликт адресов). Имейте в виду, что эти проекты предназначены только для демонстрационных образовательных целей.

3.1.2.1 Получение кадров с сообщений J1939

Этот проект Arduino стал проблемой после того, как я забыл с войти с собственным счетом.

Примечание. Для тестирования приложения CAN/J1939 вам потребуется как минимум два

Узлы CAN/J1939 для установления сетевой связи. Вторым узлом может быть другой Arduino с шилдом CAN или (если позволяет бюджет) другой устройство CAN/J1939 с возможностью мониторинга данных CAN/J1939.

Другими словами, для приема сообщений J1939 мне понадобился передатчик.

Однако решение было простым: я ис пользовал второй Arduino с шилдом CAN и загрузил приложение, как описано в главе «Пример моделирования SAE J1939», в данном случае расширенную версию.

Чтобы получать и передавать кадры с сообщений J1939 более полным образом (вместе с приемом простых сообщений CAN, которые включают декодирование идентификатора сообщения), я добавил дополнительные функции J1939 в модуль can.cpp.

Новые (или модифицированные) вызовы функций:

j1939Transmit — этот вызов функции (который мы уже ис пользовали в предыдущем проекте) теперь поддерживает тройку исх одног о и конечног о адресов — предыдущая версия допускала только исх одногий адрес и, таким образом, напрямую не поддерживала одноранговую связь.

j1939Receive — параметры этой функции очень похожи на параметры, ис пользованные с j1939Transmit, с тем отличием, что параметры являются указателями на переменные, потому что это параметры, которые мы получаем.

canReceive — эта функция (вызывающаяся j1939Receive) была расширена для ис пользования кольцевого буфера с сообщений CAN, чтобы предотвратить проблемы с синхронизацией и, следовательно, избежать потери сообщений.

j1939PeerToPeer — эта функция (вызывающаяся как j1939Transmit, так и j1939Receive) определяет, представляет ли переданный PGN обмен данными между одноранговыми узлами (прямой узел). Если да, функция вставляет адрес назначения в идентификатор сообщения CAN.

Фактическая функция loop() довольно проста, так как большая часть кода ис пользовается для отображения полученных кадров данных J1939.

```
// -----
```

```
// Основной цикл — точка входа  
Arduino // -----
```

Пустой цикл()

```

{
// Байт обявлений
nPriority; байт
nSrcAddr; байт
nDestAddr; байт
nData[8]; интервал
ндален;
длинный лПГ и;

символ sString[80];

// Проверка полученных сообщений J1939
if(j1939Receive(&IPGN, &nPriority, &nSrcAddr, &nDestAddr, nData,
    &nDataLen) == 0)
{
    sprintf(sString, "PGN: 0x%X Src: 0x%X Dest: 0x%X", (int)IPGN, nSrcAddr, nDestAddr);
    Серийный.print(sString);
    если (nDataLen == 0)

        Serial.print("Нет данных.\n\r");
    else

        { Serial.print("Данные: ");
        for(int nIndex = 0; nIndex < nDataLen; nIndex++) { sprintf(sString,
            "0x%X", nData[nIndex]); Серийный.print(sString);

        }// конец
        для Serial.print("\n\r");
    }// конец еще

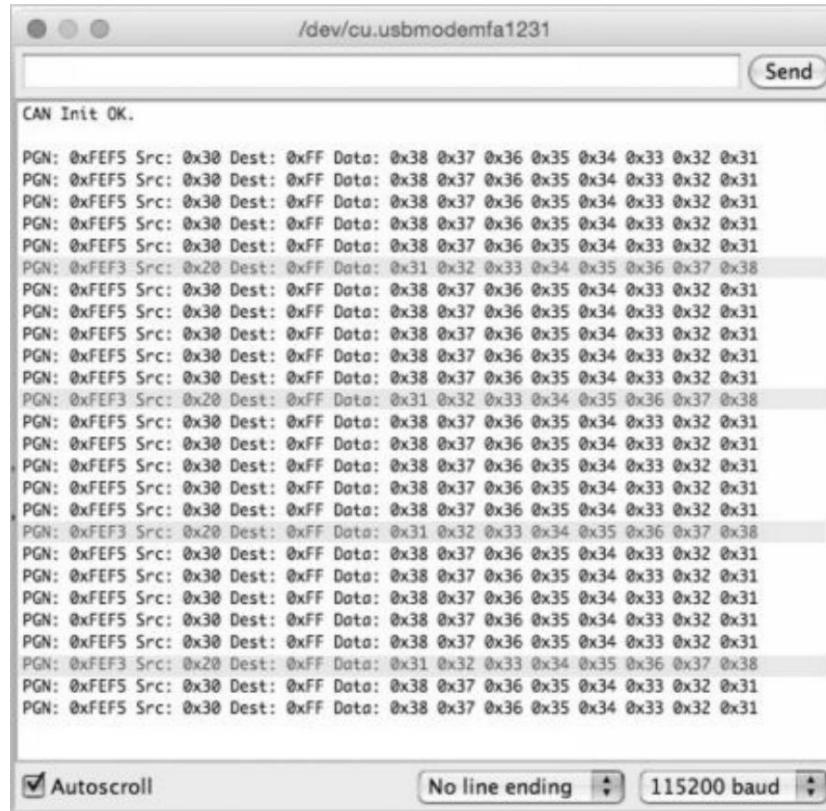
    }// конец , если

}// конец цикла

```

Примечание. Этот проект Arduino доступен на странице загрузки по адресу <http://ard1939.com>.

Данные отображаются на последовательном мониторе Arduino, и результат соответствует ожидаемому (для демонстрации выделил PGN 0xEF03, который показывает каждые пять секунд).



Примечание. Адрес назначения отображается как 0xFF (255). 255 — это глобальный адрес, означающий, что оба PGN являются широковещательными сообщениями.

3.1.2.2 Прием и ответ на кадры запроса J1939

В следующем проекте мы обнаружим еще одну функцию протокола SAE J1939, а именно с сообщение запроса (как определено в стандарте SAE J1939/21).

Как и в предыдущем проекте, нам понадобятся два узла J1939, значит, потребуется два проекта, один для получения запроса и ответа, второй для отправки запроса и получения ответа.

Но сначала давайте взглянем на сообщение запроса: этот тип сообщения, представлена в виде выделенным PGN, представляемый с предсторонней для запроса информацией глобально (широковещательно) или с определенным образом узла (одноранговая сеть).

- PGN 59904 - Сообщение о запросе за вленного адреса.
 - Номер группы параметров: 0xEAx (где xx представляет адрес назначения, либо глобальный адрес

- 255 для включения или адрес конкретного узла)
- Длина данных : 3 байта
- Данные: байты 1, 2, 3 = запрашивается PGN (с начала младший бит, MSB пос ледний)
- Приоритет по умолчанию : 6

Примечание. Передача/прием 3-байтового PGN в соответствии с стандартом SAE J1939 всегда сопровождается толку программистов, плохознакомых с J1939, в частности, отправка LSB первым и MSB последним, что противоречит мышлению обычного программиста. Что еще хуже, в масивном 1600-страничном справнике с стандартами SAE J1939 есть только одно небольшое предложение, которое относится к этой маленькой, но важной детали.

Узел, получивший сообщение запроса, ответит запрошенным PGN и его данными.

Для нашего текущего проекта мы с озадачим следующий сценарий: узел 0x20 (32) отправляет запрос на PGN 65262 (0xFFFF — температура двигателя) узлу 0x30 (48). Узел 0x30 получит запрос и ответит отправкой PGN.

Примечание. Номера узлов и запрошенный PGN выбраны с лучшим образом и с легкостью только в качестве демонстрационных примеров. Как сомнительно, метод запроса с сообщением, как показано здесь, не на 100% совместим с J1939. Один из следующих проектов, сотовой сканер J1939, является лучшим примером пользования сообщениями запроса.

Чтобы повторить поставленные задачи:

1. Узел 0x30 получает сообщение запроса и отвечает, отправляя PGN
2. Узел 0x20 отправляет сообщение запроса и получает ответ

Чтобы упростить управление проектом, я объединил две задачи в один проект и разместили его в две отдельные системы Arduino. В этом случае обе системы имитируют адреса обоих узлов.

Следующий код, выдернутый из функции `loop()`, объясняет функцию этого проекта:

```

// Проверка полученных сообщений J1939
if(j1939Receive(&IPGN, &nPriority, &nSrcAddr, &nDestAddr, nData,
    &nDataLen) == 0)
{
    // Отображаем полученное сообщение на последовательном мониторе
    :
    :

    // Анализ полученного PGN
    switch(IPGN) { case

        PGN_RequestMessage:

            if(nData[0] == PGN_EngineTemperatureLSB && nData[1] ==
                PGN_EngineTemperature2ND && nData[2] ==
                PGN_EngineTemperatureMSB) {

                // Получено сообщение с запросом температуры двигателя
                j1939Transmit(PGN_EngineTemperature, 6, NODE_Request, NODE_Response, msgEngineTemperature, 8); // передаем сообщение
            }

            перерыв;

        }// конечный переключатель

    } // конец, если

    // Проверяем пользовательский ввод для каждого последовательного монитора
    // Отправляем сообщение запроса пользователем
    вводом if(ReadSerialMonitorString() == true)
        j1939Transmit(PGN_RequestMessage, 6, NODE_Request, NODE_Response, msgRequest, 3);

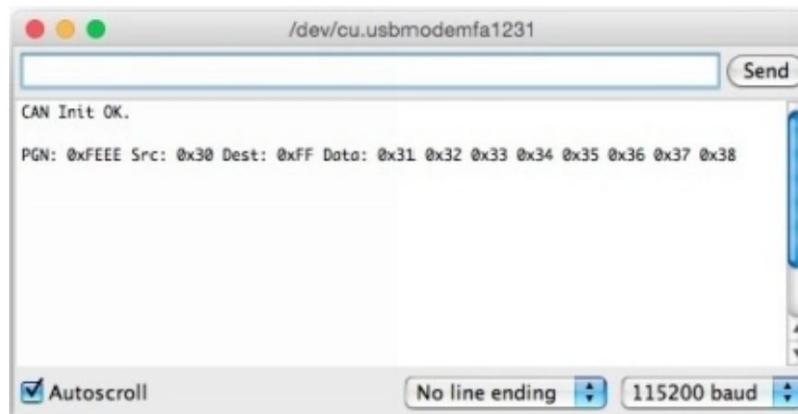
    :
    :
}

```

1. Как только мы получаем адрес сообщения J1939, мы отображаем его на последовательном мониторе (код был размыт).
2. Анализируем полученный PGN.
3. Если PGN является PGN сообщения запроса, мы проверяем прикрепленные данные для запрошенного PGN.
4. Если запрошенный PGN относится к температуре двигателя, мы передаем соответствующий PGN.
5. Если пользователь отправляет какие-либо данные через последовательный монитор, мы отправляем запрос на температуру двигателя.

Примечание. Этот проект Arduino доступен на странице загрузки по адресу <http://ard1939.com>.

На следующем изображении показан последовательный монитор Arduino после того, как пользователь отправил сообщение с запросом. Другой узел Arduino ответил на запрос.



Обратите внимание, что адрес назначения отображается как 0xFF (255), что является глобальным адресом. Как я упоминал ранее, этот пример не совместим с J1939.

Адрес назначения должен быть глобальным адресом, потому что PGN 65262 (0xFEEE) не может быть передан от одноранговой сети, а только как широковещательное сообщение.

На следующем скриншоте экрана, сделанном с помощью анализатора CAN ADFweb, показан трафик данных по автомобильной шине J1939, что подтверждает функциональность проекта.

NR	TIME	ID (HEX)	DATA (HEX)	ASCII
1	12:52:11 AM.672.0	18EA3020	EE FE 00	ip
2	12:52:11 AM.674.0	18FEEE30	31 32 33 34 35 36 37 38	12345678

Линия 1

- 0x18 указывает на приоритет сообщения 6.
- 0xEA30 указывает на сообщение запроса, отправленное на адрес узла 0x30.
- 0x20 представляет собой адрес запрашивающего узла (адрес источника).

Строка 2:

- 0x18 представляет собой приоритет сообщения
- 6. 0xFEEE — это PGN для температуры двигателя (по запросу). 0x30
- представляет собой адрес передающего узла.

3.1.3 Простой SAE J1939 на USB-шлюз

Считаю необх одимым дать определение «SAE J1939 to USB Gateway», пос кольку сущес твует не с колько вариантов применения шлюза.

Если вы просматриваете Интернет в поисках шлюзов J1939, вы в первую очередь найдете шлюзы CAN, которые поддерживают преобразование 11-битных и 29-битных сообщений CAN в другую последовательную технологию, такую как USB, RS232 и т. д. Поскольку SAE J1939 основан на сообщения CAN с 29-битным идентификатором, многие производители позволяют себе называть свои устройства «шлюзами SAE J1939», что не только вводит в заблуждение, но и может привести к техническим проблемам при подключении этих устройств к реальному транспортному средству (дизельному двигателю).

Проблема заключается в том, что эти так называемые шлюзы не предоставляют ясно все функции протоколов SAE J1939 и, следовательно, не могут поддерживать процесс запроса адресов в соответствии с SAE J1939/81. При передаче данных в автомобильную сеть J1939 вам необх одимо предоставить адрес узла (источника) с кадром данных, и этот адрес узла должен быть уникальным. Отсутствие ясного протокола в шлюзе может привести к конфликтам идентификаторов узлов и, следовательно, вызвать серьезные технические проблемы.

Эту проблему можно обойти, если все идентификаторы узлов в сети ясно закодированы, и вы назначаете свободный идентификатор узла сообщениям, отправляемым в сеть.

Это может работать в ограниченных случаях, но этот метод естественно, не рекомендуется.

Назначением этих типов шлюзов J1939 должен быть простой мониторинг данных, т.е. только получение сообщений J1939 (вам необх одимо расшифровывать сообщения J1939 длиннее 8 байт, чтобы вляется трудоемкой задачей). Для связи сшиной транспортного средства (двунаправленная связь) необх одима полная реализация протокола SAE J1939 с процедурой утверждения адреса и поддержкой сообщений длиннее 8 байт.

Тем не менее, в дальнейшем я представлю этот вид шлюза с ясно обозначенными недостатками, но я делаю это исключительно в демонстрационных и образовательных целях.

Эскиз Arduino, представленный ниже, является первым шагом к полноценному шлюзу SAE J1939 с ясно обозначенным ясным протоколом (см. главу ARD1939 — Стандартные протоколы SAE J1939 для Arduino).

Некоторые из предыдущих эскизов Arduino, представленных в этой книге, уже представляют функции для просмотра шлюза SAE J1939-USB. Однако они

были предназначены только для приема с сообщений. Приложение шлюза требует, чтобы мы также могли отправлять с сообщения. Необходимо двунаправленной связи также приводит к следующему требованию к шлюзу J1939, а именно к определению протокола связи между шлюзом и хостом темой (обычно ПК).

Поскольку наше внимание сосредоточено на оборудовании Arduino, соединение между шлюзом (то есть Arduino) и ПК устанавливается через интерфейс USB. Основа протокола USB уже установлена с помощью функции Serial.print, и протокол будет основан на передаче текста ASCII.

Обмен данными осуществляется через последовательный монитор Arduino, но также представлю регламенту последовательного монитора с Visual Studio C# в следующий раз. Если вы хотите создать более сложную передачу данных между Arduino и ПК, следующие проекты обеспечивают прочную основу.

Наш простой шлюз J1939-USB будет выполнять три функции:

- Сетевой сканер J1939, то есть Arduino, будет сканировать есть J1939 на наличие существующих узлов и извлекать из них информацию.
- Монитор трафика данных, аналогичный ранее представленным проектам.
- Симулятор трафика данных, то есть Arduino, будет отправлять в есть определяемые пользователем сообщения.

Примечание. Как обычно, следующие проекты Arduino доступны на странице загрузки по адресу <http://ard1939.com>. Однако все три функции (сканер сети, монитор трафика данных и симулятор трафика данных) объединены в один проект, поскольку две лестничные функции включаются дополнениями к предыдущему проекту.

3.1.3.1 Сетевой сканер J1939

Как и в предыдущем проекте (см. раздел «Прием и ответ на кадры запроса J1939»), мы будем использовать сообщение запроса SAE J1939, в данном случае для запроса адресов узлов из сети транспортного средства. Сообщение запроса (также известное как сообщение запроса захваченного адреса) также используется в процессе запроса адреса, как определено в SAE J1939/81. Наш проект сетевого сканера очень похож на процесс с утверждениями адреса, поскольку мы запрашиваем адреса из сети.

(Процесс с заявки на адрес будет запрашивать конкретные адреса, чтобы узнать, заняты они или нет).

Примечание. Опять же, этот пример проекта будет работать только с двумя узлами: один Arduino загружен с последующим проектом, а второй узел загружен полным текущим протоколов ARD1939 — SAE J1939 для Arduino (см. также главу ARD1939 — Реализация).

Еще раз давайте взглянем на сообщение Request for Address Claimed:

- PGN 59904 - Сообщение о запросе заявленного адреса
 - Номер группы параметров: 0xEAx (где xx предстает адрес назначения, либо глобальный адрес 255 для широковещательной рассылки, либо адрес конкретного узла)
 - Длина данных : 3 байта
 - Данные: байты 1, 2, 3 = запрошенный PGN (с начальномладший бит, последний старший бит)
 - Приоритет по умолчанию : 6

Сообщение Request for Address Claimed (при использовании полного текущего протокола с его помощью) используется для запроса отправки сообщения Address Claimed либо с определенного узла в сети, либо со всех узлов (использование глобального адреса назначения = 255). Сообщение Address Claimed (как описано ниже) предоставляет запрошеннную информацию, т. е. адрес и НАЗВАНИЕ отвечающего узла(узлов).

Целью отправки такого запроса может быть несколько причин, например:

- Узел проверяет, может ли он претендовать на определенный адрес .
- Узел проверяет наличие другого узла (приложения -контроллера) с определенной функцией.

Ответ на сообщение Request for Address Claimed может быть несколькими:

- Любой адресуемый узел, который уже заявил адрес , ответит с сообщением Address Claimed .
- Любой адресуемый узел, который не мог запросить адрес , ответит с сообщением Cannot Claim Address .
- Любой адресуемый узел, который еще не заявил адрес , должен сделать это.

ответив с вом с обс твенным с обще нием Address Claimed , г де ис х одный адрес имеет значение NULL (254).

- Узел, отправля ю щий с обще нием Request for Address Claimed, должен ответить на свой с обственныи запрос в с лу чае, ес ли был ис пользован г лобальныи адрес назначения (255).

Ответом на с обще нием Request for Address Claimed я вля етс я фактичес кий адрес .

Зая вленное с обще нием, как определено в с ледую щем:

- PGN 60928 - с обще нием с зая вленным адрес ом
 - Номер г руппы параметров: 0xEExx (г де xx пред ставля ет адрес назначения , либо г лобальныи адрес 255 для широк овещательной передачи, либо адрес конкретног о узла)
 - Длина данных : 8 байт
 - Данные: НАЗВАНИЕ управля ю щег о приложения (с м. также г лаву SAE J1939/81 — Проц едура обращения с адрес ом
 - Приоритет по умолчанию : 6

Код для нашег отекущег о проекта Arduino, Network Scanner, огя ть же довольно прос т:

```
недействительный ц икл
0{
// Байт
объявлений nPriority
= 0; длинный
IPGN = 0; байт nSrcAddr
= 0; байт nDestAddr =
0; байт nData[8];
интервал ндатален;

// Отправить запрос на зая вленный адрес нажатием клавиши
if(Serial.available() > 0)

{ Serial.setTimeout(1);
Serial.readBytes (sString, 1);
Serial.print("Иниц ировано с канирование с ети:
\n\r"); nData[0] = PGN_AddressClaimedLSB;
nData[1] = PGN_AddressClaimed2ND; nData[2]
= PGN_AddressClaimedMSB; j1939Transmit
(PGN_RequestMessage,
PRIORITY_RequestMessage, SA,
GLOBALADDRESS, nData, 3);
}
```

```

// Проверка полученных сообщений J1939
if(j1939Receive(&IPGN, &nPriority, &nSrcAddr, &nDestAddr, nData,
&nDataLen) == 0)

{ switch(IPGN) { case

PGN_AddressClaimed: sprintf(sString, "Addr
Claimed: 0x%X %dd\n\r", nSrcAddr, nSrcAddr); Сериалный.print(sString); перерыв;

// Мы добавим сюда больше кода для следующего проекта...

}// конечный переключатель

}// конец , если

}// конец цикла

```

- Как только пользователь вводит ввод через последовательный монитор, мы передаем сообщение Request for Address Claimed . Мы используем глобальный адрес (255) для получения ответов от всех узлов сети.
- Мы проверяем интерфейс CAN на наличие полученного кадра данных J1939.
- Если сообщение было получено, мы проверяем, было ли это сообщением Address Claimed .
- Если сообщение было с сообщением Address Claimed , мы печатаем полученный адрес .

Моя конфигурация позволяет мне подключить к сети два узла SAE J1939, систему ARM по адресу 0x2b (43) и Arduino Mega 2560 по адресу 0x80 (128).

Следующий скриншот экрана с бортового монитора подтверждает конфигурацию и адреса:



В качестве еще одного доказательства концепции давайте еще раз взглянем на программу-анализатор ADFweb, которую я все еще подключал:

NR	TIME	ID (HEX)	DATA (HEX)
1	12:11:56 AM.134.2	18EAFF33	00 EE 00
2	12:11:56 AM.134.8	18EEFF2B	00 00 60 43 00 FF FE 90
3	12:11:56 AM.135.5	18EEFF80	00 00 E0 FF 00 FF FE 10

- Стока 1: наше приложение Arduino отправляет сообщение Request for Address Claimed . Обратите внимание на идентификатор, где 18 указывает на приоритет 6, EAFF предстает с собой с сообщение Request for Address Claimed , включая глобальный адрес 255 (0xFF). Последний байт 0x33 предстает с моделированный исходный адрес нашего приложения . Поле данных содержит трехбайтовый PGN для сообщения Заявленный адрес (00E000; с начала LSB, последним MSB).
- Стока 2: Узел с адресом 0x2B (43) отвечает с сообщением Address Claimed . Обратите внимание на идентификатор, где 18 указывает на приоритет 6, а EEFF предстает с собой с сообщение Address Claimed , включая глобальный адрес 255 (0xFF). Последний байт предстает исходный адрес узла. Поле данных содержит восемьбайтовое НАЗВАНИЕ управляющую шеф приложения (ECU).
- Стока 3: То же, что и строка 2, но с другим адресом и ИМЯ.

На следующем шаге, чтобы продемонстрировать различия между широковещательной рассылкой с сообщений и одноранговой сетью, я заменил GLOBALADDRESS в функции j1939Transmit на

0x80, а это значит, что мы проверяем есть ли наличие одного конкретного адреса.

В конце концов, результат на последовательном мониторе Arduino не очень впечатляет; он показывает (как и ожидалось) только один ответ:



Однако программное обеспечение анализатора CAN ADFweb раскрывает более подробную информацию о одноранговой связи (от узла к узлу):

NR	TIME	ID (HEX)	DATA (HEX)
1	12:37:08 AM.949.0	18EA8033	00 EE 00
2	12:37:08 AM.950.1	18EE3380	00 00 E0 FF 00 FF FE 10

- Стока 1: наше приложение Arduino отправляет сообщение Request for Address Claimed . Обратите внимание на идентификатор, где 18 указывает на приоритет 6, EAFF предстает с сообщением «Запрос заявленного адреса», включая запрошенный адрес 0x80 (128). Последний байт 0x33 предстает с моделированный исходный адрес нашим приложением . Поле данных содержит трехбайтовый PGN для сообщения «Заявленный адрес» (00E000, LSB первым, MSB последним).
- Стока 2: Узел с адресом 0x80 (128) отвечает с сообщением Address Claimed . Обратите внимание на идентификатор, где 18 указывает на приоритет 6, EE33 предстает с сообщением «Заявленный адрес», включая адрес запрашивающей стороны 0x33. Последний байт предстает исходный адрес узла. Поле данных содержит восемьбитное НАЗВАНИЕ управляющей юниверсальной единицы (ECU).

3.1.3.2 Мониторинг трафика данных J1939

В следую щем с кетче Arduino «мониторинг трафика данных » означает не что иное, как получение и отображение с общефнных данных J1939 (с м. также главу « Получение кадров с общефнных J1939»).

Вмес тог о, чтобы с оздавать новый проект, нам нужно вс ег олишь добавить не сколько строк кода к предыду щему проекту с етевог ос канера, в час тнос ти раздел switch(IPGN) (с м. выделенный код):

```
switch(IPGN) { case

    PGN_AddressClaimed: sprintf(sString,
        "Addr Claimed: 0x%X %dd NAME: ", nSrcAddr, nSrcAddr); Серийный.print(sString); for(int nIndex = 0;
        nIndex < nDataLen; nIndex+
    +) { sprintf(sString, "0x%X", nData[nIndex]); Серийный.print(sString); }

    Serial.print("\n\r");
    перерыв;

    по
    умолчанию: sprintf(sString, "PGN: 0x%X Src: 0x%X Dest: 0x%X", (int)IPGN, nSrcAddr, nDestAddr);
        Серийный.print(sString); если
        (nDataLen == 0 )

    Serial.print("Нет данных .\n\r"); else

    { Серийный.print("Данные: "); for(int
        nIndex = 0; nIndex < nDataLen; nIndex++) { sprintf(sString, "0x%X",
        nData[nIndex]); Серийный.print(sString); }

    Серийный.print("\n\r");

    }// конец еще

    перерыв;

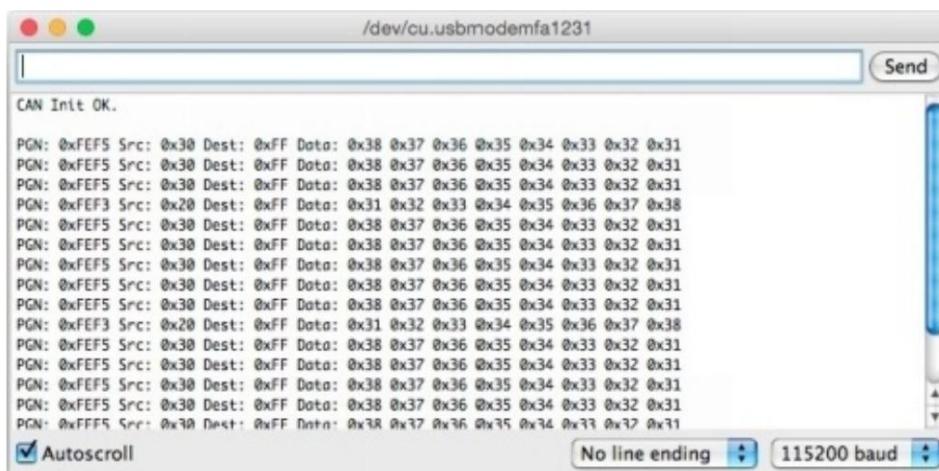
}// конечный переключатель
```

После фильтрации с общефния Address Claimed мы также разрешаем любые другие PGN, которые мы получаем от автомобильной шины. В этом случае мы отображаем не только PGN, но и адрес источника (передаю щий узел), адрес получателя (получаю щий узел) и фактические данные.

Примечание. Для вариантов этого проекта вы можете фильтровать дополнительные, определенные PGN или даже фильтровать с сообщения, например, по адресу назначения (где адрес назначения — это адрес этого управляющего приложения).

Чтобы протестировать с кетч, нам снова понадобится еще один узел J1939, то есть еще один Arduino с экраном CAN для создания трафика данных. В следующем примере я исользовал проект, как описано в главе Пример моделирования SAE J1939.

Результат такой же, как и ожидалось:



3.1.3.3 Моделирование трафика данных J1939

В предыдущем проекте (см. главу Пример моделирования SAE J1939) я продемонстрировал моделирование трафика данных SAE J1939. Однако с сообщения в этом примере передавались с определенной частотой. Далее мы будем передавать с сообщения в соответствии с вводом пользователя.

Как и прежде, мы будем использовать предыдущий скрипч и добавлять код, не удаляя ранее добавленный функционал. Но, поскольку мы будем передавать данные в соответствии с пользовательским вводом, нам необходимо отдельить сетевой канер от передачи данных (Помните: сканирование сети инициировалось любым пользовательским вводом).

Из с кетча предыду ще й г лавы я удалил (точнее, закомментировал) с ледую щий код:

```

если (Serial.available () > 0)

{ Serial.setTimeout (1);
  Serial.readBytes (sString, 1);
  Serial.print ("Иницировано сканирование с эти:\n\r");
  nData[0] = PGN_AddressClaimedLSB; nData[1] =
  PGN_AddressClaimed2ND; nData[2] =
  PGN_AddressClaimedMSB; j1939Transmit
  (PGN_RequestMessage, PRIORITY_RequestMessage, SA,
   ГЛОБАЛЬНЫЙ АДРЕС, nData, 3);

// конец , если

```

Новый скетч теперь содержит вызов функции ReadSerialMonitorString , которая возвращает текстовую строку, введенную пользователем, плюс длину записи. Для сканирования с этим я выбрал «s» в качестве команды. Следовательно, код выглядит так:

```

// Проверка пользователя о ввода для монитора
последовательного порта int nCount =

ReadSerialMonitorString(sString); if(nCount > 0)

{ if(nCount == 1 && sString[0] == 's')
  { Serial.print ("Иницировано сканирование с эти:
\n\r"); nData[0] = PGN_AddressClaimedLSB; nData[1]
  = PGN_AddressClaimed2ND; nData[2] =
  PGN_AddressClaimedMSB; j1939Transmit (PGN_RequestMessage, PRIORITY_RequestMessage, SA,
   GLOBALADDRESS, nData, 3);

// конец , если
:
: Больше кода для подражания здесь...
:
}

// конец , если

```

На следующем скриншоте экрана показан тот же результат, что и в главе Сетевой сканер J1939, но для добавления дополнительной информации я добавил код для печати ИМЯ управляющего щегла приложения .



Как я упоминал ранее, для реализации новых функций шлюза с помощью Arduino нам потребуется установить протокол связи между USB-интерфейсом Arduino и ПК. Основа этого USB-протокола уже заложена с помощью функции `Serial.print`, и протокол будет основан на передаче текста ASCII.

Кроме того, нам нужно указать команды для отправки данных J1939 в соответствии с вводом пользователя. Как описано ранее, мы используем букву «s», чтобы инициализировать сеть.

сканирование.

Для отправки полного кадра данных SAE J1939 я определил следующий формат пользователя (когда вводится текст ASCII):

PGN – 4 символа

Приоритет – 1 символ

Исходный адрес – 2 символа

Адрес назначения – 2 символа

Данные – от 2 до 16 символов (от 1 до 8 байтов данных)

Все параметры будут разделены пробелом, и предполагается, что все параметры вводятся в шестнадцатеричном формате.

Пример: EAFF 6 33 FF 00EE00

В приведенном выше примере (с сообщение о заявлении о глобальном адресе) мы отправляем PGN 0xEAFF с приоритетом 6, исходным адресом 0x33, глобальным

адрес назначения (255) и три байта данных .

Примечание. Следующий код выполняет лишь очень элементарную проверку пользователя с кодом ввода. Пожалуйста, не стесняйтесь улучшать код или добавлять дополнительные функции в протокол. Дополнительными функциями могут быть, например, добавление/удаление фильтров PGN (разрешение только определенных пользователем PGN).

Показанный код является расширением предыдущего фрагмента кода (см. выделенный раздел).

```

int nCount = ReadSerialMonitorString(sString); if(nCount
> 0) { if(nCount
== 1 && sString[0] == 's')

{ Serial.print("Иницировано сканирование с эти:
\n\r"); nData[0] = PGN_AddressClaimedLSB;
nData[1] = PGN_AddressClaimed2ND; nData[2]
= PGN_AddressClaimedMSB; j1939Transmit
(PGN_RequestMessage, PRIORITY_RequestMessage, SA, GLOBALADDRESS, nData, 3);

}// конец ,
если else if(ParseUserCommand(sString, nCount, &IPGN, &nPriority, &nSrcAddr,
&nDestAddr, nData, &nDataLen) == true)

{ // Выводим отзыв для проверки sprintf(sString, "Передача
PGN: 0x%XP: 0x%X SA: 0x%X DA: 0x%X Data: ", (int)IPGN, nPriority, nSrcAddr, nDestAddr);
Серийный.print(sString); for(int nIndex = 0; nIndex <
nDataLen; nIndex++)
{ sprintf(sString, "0x%X", nData[nIndex]);

Серийный.print(sString); }// конец для
Serial.print("\n\r\n\r");

// Передаем сообщение J1939
j1939Transmit(IPGN, nPriority, nSrcAddr, nDestAddr, nData, nDataLen);

}// end else if
else

{ // Неверный пользовательский ввод; вывести соответствующее
с сообщение Serial.print("Неверный ввод
данных :");
Серийный.print(sString); Serial.print("\n\r\n\r");

}// конец еще

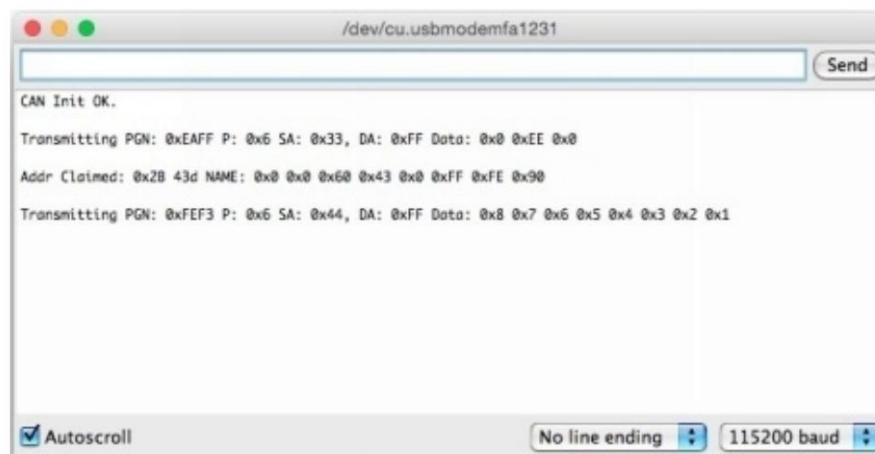
```

```
// конец , если
```

Вы можете заметить новую функцию ParseUserCommand, которая анализирует ввод пользователя и возвращает результат (true = правильно, false = неправильно). Когда вх одные данные признаны правильными, функция возвращает PGN, приоритет, адрес источника (SA), адрес назначения (DA) и фактические данные.

В качестве обратной связи ссылаются на пользователя сообщение J1939 печатается на последовательном мониторе, а затем передается в сеть J1939.

На следующем снимке экрана показан последовательный монитор, и в качестве доказательства концепции я ввел два сообщения.



Первое сообщение — это сообщение Request for Address Claimed, поэтому последовательный монитор показывает реакцию сети (адрес узла 0x2B). Второе сообщение — это сообщение о местоположении автомобиля (PGN 65267 = 0xEF03), которое мы использовали в предыдущем примере (см. главу «Пример моделирования SAE J1939»).

Следующий снимок экрана программы обес печения анализатора CAN ADFweb подтверждает функциональность:

NR	TIME	ID (HEX)	DATA (HEX)
1	12:42:53 AM.894.5	17EAFF33	00 EE 00
2	12:42:53 AM.895.1	18EEFF2B	00 00 60 43 00 FF FE 90
3	12:46:18 AM.771.4	17FEF344	08 07 06 05 04 03 02 01

- Стока 1: Arduino отправляет сообщение Request for Address Claimed .
- Стока 2: узел SAE J1939 по адресу 0x2B отвечает с сообщением Address Claimed .
- Стока 3: Arduino отправляет сообщение о местоположении автомобиля (PGN 65267).

Как я упоминал ранее, хотя этот проект является лишь простым примером, существоует множество возможностей расширения кода до профессионального решения SAE J1939. Однако важной частью шлюза SAE J1939-USB является визуализация полученных и переданных данных. Следующий код даст вам основу для создания профессионального графического пользовательского интерфейса под Windows.

3.1.3.4 Код Visual Studio

В то время как программирование Arduino может быть захватывающим (тем более, что все работает так гладко), настиче удовольствие приходит, когда вы можете распространить возможности Arduino до ПК под управлением Windows.

Примечание. Приношу свои извинения всем пользователям Mac и LINUX за пример программирования для Windows, но нет лучшего программирования, чем использование C# в Microsoft Visual Studio. Мне нравилось программировать под OS-X и LINUX, но когда дело доходит до создания быстрых и эффективных примеров программирования, я предпочитаю оставаться в Visual Studio. Однако опытный программист должен быть в состоянии воспроизвести функциональность последовательного монитора.

Чтобы узнать больше о программировании последовательных портов (RS-232 и USB) в LINUX, см. <http://www.teuniz.net/RS-232/>. Я считаю это самым профессиональным приложением для последовательных портов под LINUX. Он также подходит для приложений Windows, но в первую очередь предназначен для компиляторов ниже Visual Studio и/или для программирования встроенных систем.

Далее мы предполагаем, что на вашем компьютере с Windows установлен USB-драйвер Arduino. Драйвер автоматически устанавливается вместе с средой разработки Arduino.

Как я уже упоминал в своей заметке, я использую Visual Studio 2012 от Microsoft и разработал следующий графический интерфейс, который может показаться разработчику Arduino очень знакомым. Поступи, эта очень простая программа является копией последовательного монитора Arduino.

Примечание. Следующий пример программирования позволит вам создать собственный графический интерфейс пользователя. Он демонстрирует все новые, например, как считывать данные с USB-порта ПК, записывать данные и отображать данные на экране.



Элементы экрана представляют собой текстовое поле для ввода данных, командную кнопку для отправки ввода в Arduino и еще одно текстовое поле большего размера для отображения данных, полученных от Arduino. Наконец, что не менее важно, есть поле со списком, отображающее все доступные USB COM-порты (ваша задача — определить правильный USB-порт; автоматическое определение отсутствует).

Чего программа не предоставляет, так это настройки скорости передачи данных, которые были жестко задокументированы в программу как 115 200 бод, но могут быть легко изменены. Конечно, это не обязательно профессиональный способ сделать это, но, в концах концов, этот пример программирования служит примером чтения / отправки сообщений из/в Arduino.

Что касается более професионального инструмента мониторинга и диагностики, вам потребуется я больше различных элементов экрана, и настройка с вопросами передачи данных должна быть частью этого проекта. Поэтому не стесняйтесь добавлять дополнительные функции по своему усмотрению.

Все элементы экрана в этом проекте остаются со своими настройками по умолчанию, однако за некоторыми исключениями, как показано ниже:

Элемент	Имя	Измененное свойство
События		
Форма	Form1	Текст = «Последовательный монитор»
Текст	txtОтправить	
Текст	txtReceived	Многстрочный = Истина Полосы прокрутки = вертикальные
Кнопка	btnОтправить	Текст = «Отправить»
Нажмите		
Поле со списком	cboCOMPort	
Селектединдекс чанжед		

Ниже показан листинг программы C# (все программы находятся в форме):

```
с помощью
ис темы; используя System.Collections.Generic;
ис используя System.ComponentModel;
ис используя System.Data; с
помощью System.Drawing; с
помощью System.Linq;
ис используя System.Text;
ис пользование System.Threading.Tasks;
ис используя System.Windows.Forms; с
помощью System.IO;
ис пользование System.IO.Ports;
ис пользование System.Threading;

пространство имен USBAccess

{ общедоступный частичный класс Form1 : Form
{
    // Константы

    public const int REC_BUFFER_SIZE = 500; общественное
    константное число READ_TIMEOUT = 500; публичный
    интервал WRITE_TIMEOUT = 500;
```

```

общедоступная константа REC_BUFFER_FILLTIME = 80;

общедоступный статический SerialPort _serialport;

общедоступная форма1
{
    ИнициализироватьКомпонент();

    строка[] sPorts = новая строка[20]; sPorts =
    SerialPort.GetPortNames();

    for (int nIndex = 0; nIndex < sPorts.Length; nIndex++)
        cboCOMPort.Items.Add(sPorts[nIndex]);

}// конец формы1

// SetTextDeleg //
----- частный
делегат void SetTextDeleg(string text);

// sp_DataReceived //
----- void
sp_DataReceived (отправитель объекта, SerialDataReceivedEventArgs e) {

    // Установить размер приемника
    буфер char[] sRecData = new char[REC_BUFFER_SIZE + 1];

    // Даем оборудованию некоторое время, чтобы получить все сообщение
    Thread.Sleep(REC_BUFFER_FILLTIME);

    {
        int nBytes = _serialport.BytesToRead;

        // Читаем строку int
        nIndex;

        for (nIndex = 0; nIndex < nBytes; nIndex++) {

            int nRec = _serialport.ReadByte(); sRecData[nIndex]
            = (char)nRec;

        }// конец для

        sRecData[nIndex] = (char)0; // Завершить строку string sStr = new
        string(sRecData);

        // В случае RS232 эта строка вызывает тайм-аут, // означающий,
        что данные не принимаются
        this.BeginInvoke (новый SetTextDeleg (si_DataReceived), новый объект []
        { sStr });
    }
}

```

```

} поймать (TimeoutException) { }

}// конец _serialport_DataReceived

// si_DataReceived //
----- private
void si_DataReceived(string data) {

    если (txtReceived.TextLength == 0)
        txtReceived.Text = данные;

    иначе txtReceived.Text += "\n\r" + данные;

    // Установить курсор в конец
    экрана txtReceived.SelectionStart = txtReceived.TextLength;
    txtReceived.ScrollToCaret();
    txtReceived.Обновить();

}// конец si_DataReceived

// btnSend_Click //
----- private
void btnSend_Click(отправитель объекта, EventArgs e) {

    // Убедитесь, что последовательный порт открыт перед попыткой записи
    try
    {
        если (_serialport.IsOpen)
            _serialport.Open();

        если (txtSend.Text.Length > 0)
            _serialport.Write (txtSend.Text); иначе

            MessageBox.Show("Пожалуйста, введите сообщение для отправки.",
                           "Внимание!");

    } поймать (Исключение
    ex) {
        MessageBox.Show("Ошибка открытия /записи в последовательный
                        порт." + ex.Message, "Ошибка!");
    }

}// конец btnSend_Click

// Событие:

cboCOMPort_SelectedIndexChanged //----- закрытое void cboCOMPort_SelectedIndexChanged (отправитель объекта,
                           EventArgs e)
{
    // Определяем последовательный порт для USB-
    устройство _serialport = new SerialPort(cboCOMPort.SelectedItem.ToString(),

```

```
        115200, Parity.None, 8, StopBits.One);  
        _serialport.Handshake = Рукопожатие.Нет;  
  
        // Установить время ожидания  
        чтения /записи _serialport.ReadTimeout =  
        READ_TIMEOUT; _serialport.WriteTimeout =  
        WRITE_TIMEOUT; _serialport.ReadBufferSize =  
        REC_BUFFER_SIZE; _serialport.Открыть();  
  
        _serialport.DataReceived +=  
            новый SerialDataReceivedEventHandler (sp_DataReceived);  
  
    }// конец cboCOMPort_SelectedIndexChanged  
  
}// конец класса  
  
}// конец пространства имен
```

Ссылка: Обращение с USB-портом основано на статье Райана Алфорда (с добавленным контентом Арджуна Валмики, Грегори Кшивоши и Махеша Чанда) по адресу: <http://www.c-sharpcorner.com/uploadfile/eclipsed4utoo/communicating-串行波特ом в C-Sharp/>

При запуске программы пользователю сначала необходимо выбрать соответствующий порт USB COM, который инициализирует порт (с событием SelectedIndexChanged).

Кроме того, программа работает как простой USB-терминал: сообщения вводятся сверху в текстовом поле и отправляются нажатием командной кнопки «Отправить». В большем текстовом поле отображаются полученные данные.

Ниже показан скриншот экрана, сделанный во время сеанса отправки данных J1939:



Примечание. Как обычно, этот программный проект доступен на странице загрузки по адресу <http://ard1939.com>.

3.2 ARD1939 — стек протоколов J1939 для Arduino

Самый интересный проект в этой книге — это, конечно же, полнофункциональный стек протоколов SAE J1939 для Arduino, ARD1939. ARD1939 поддерживает полный протокол SAE J1939/21 (управление сетью, процессы запроса адресов) и SAE J1939/81 (трансポートный протокол, передача сообщений размером до 1785 байт, включая сеансы ВАМ и RTS/CTS).

Техническое описание SAE J1939/21 и SAE J1939/81 см. в главе Два элемента стека протоколов SAE J1939.

Примечание. Хотя полная документация по стандарту SAE J1939 выходит за рамки этой книги, я рассмотрел обязательные основы в предыдущей главе.

Для получения более подробной информации обратитесь к «Понятному руководству по J1939», как указано в приложении к рекомендованной литературе.

3.2.1 ARD1939 — обзор функций

Программирование каждой последовательной связи (например, RS232, CAN, Ethernet, и это лишь некоторые из них) должен следовать очень простой последовательности:

1. Инициализация.
2. Чтение данных
3. Запись данных
4. Проверка состояния

Эти четыре вызова функций должны быть всем, что нужно программисту для доступа к последовательному протоколу, и нет причин, почему реализация стека протоколов SAE J1939 не должна следовать той же схеме. Копаться в ложном коде, чтобы понять, как получить доступ к функциям протокола, — просто пустая трата времени.

К сожалению, большинство коммерческих доступных стеков протоколов отличаются способностью и/или отсутствием документации. Чтобы упомянуть об этом заранее, стек протоколов ARD1939 требует очень мало вызовов функций, таким образом, позволяя программисту ввести протокол в действие за очень короткое время.

Функции, доступные для прикладного уровня SAE J1939 (т.е. вашей программы)

Я ВЛЯЮТСЯ :

Инициализация

- j1939.Init — инициализирует настройки протоколов.
- j1939.SetPreferredAddress — устанавливает предпочтительный адрес узла (источника).
- j1939.SetNAME — устанавливает ИМЯ ЭБУ с использованием отдельных параметров.
-

Чтение/запись — проверка состояния

- j1939.Operate — обрабатывает процессы утверждения адреса, считывает PGN из сети автомобиля и передает текущий статус протокола (выполняется утверждение адреса, успешное утверждение адреса, неудавшееся утверждение адреса) j1939.Transmit —
- передает данные в сеть автомобиля и обрабатывает Транспортный протокол (TP)

Другие функции приложения

- j1939.Terminate — сбрасывает настройки протоколов.
- j1939.GetSourceAddress — предоставляет собственный адрес узла.
- j1939.DeleteMessageFilter — удаляет фильтр сообщений.

Подробное описание вызовов этих функций см. в приложении «Справочник по стеку протоколов ARD1939».

3.2.2 ARD1939 — Реализация

Я предоставляю ARD1939 (стек протоколов SAE J1939 для Arduino) «как есть», т.е. я предоставляю весь проект (Arduino Sketch). Чтобы скопировать и настроить проект, используйте функцию «Создать как» в IDE Arduino.

Я думал о создании библиотеки Arduino, но считаю процесс сборки (подключение вашего проекта к библиотеке, интеграцию функциональности MCP2515 и функций интерфейса CAN и т. д.) с лишним. Полный проект, который я предоставляю, не только сэкономит ваше время, но также является проверенным решением, таким образом,

не приводить к бесконечным исключениям ошибок.

Примечание. Напоминаем, что ограничения ресурсы памяти Uno допускают максимальную длину данных J1939 только 256, в то время как версия Mega поддерживает полные 1785 байтов на каждый сообщение. Всего обе версии идентичны по своим функциональным возможностям.

3.2.2.1 Наследство функциональности

В целом скетч поддерживает три версии с текущими протоколами ARD1939: две для Arduino Uno (ARD1939-Uno и ARD1939-Uno/TP) и одну для Arduino Mega 2560 (ARD1939-Mega).

1. ARD1939-Uno — в целях экономии ресурсов памяти эта версия не поддерживает транспортный протокол (TP) в соответствии с SAE J1939/21.

Место для хранения программ: ~8200 байт (~25%)

Динамическая память: ~510 байт (~25%)

Фильтры с сообщений:

- 10.2. ARD1939-Uno/TP — эта версия поддерживает транспортный протокол (TP), но только для сообщений длиной до 256 байт.

Место для хранения программ: ~12 000 байт (~37%)

Динамическая память: ~1100 байт (~55%)

Фильтры с сообщений:

- 10.3. ARD1939-Mega — эта версия поддерживает полный текущий протоколов SAE J1939, включая транспортный протокол (TP) для сообщений длиной до 1785 байт в соответствии с стандартом J1939.

Место для хранения программ: ~12 300 байт (~4%)

Динамическая память: ~4600 байт (~56%)

Примечание. Точное представление для хранения программ и требования к динамической памяти также зависят от расширения фактического приложения J1939 в функции цикла().

Наследство функциональности можно найти в файле ARD1939.h, который является частью проекта.

```
// Версия
// программы // -----
// 0 - ARD1939-Uno
// 1 - ARD1939-Uno/TP
```

```
// 2 - ARD1939-Mega
#define ARD1939VERSION 2

// Нас тройки
J1939 #if ARD1939VERSION
    == 0 #define TRANSPORT_PROTOCOL 0
    #define J1939_MSGLEN
    #define MSGFILTERS 8 10
#endif

#if ARD1939VERSION == 1
    #define TRANSPORT_PROTOCOL 1
    #define J1939_MSGLEN 256
    #define MSGFILTERS 10
#endif

#if ARD1939VERSION == 2
    #define TRANSPORT_PROTOCOL 1
    #define J1939_MSGLEN 1785
    #define MSGFILTERS 100
#endif
```

Версия программы (ARD1939VERSION) позволяет активировать транспортный протокол (TP), где 1 = активен и 0 = неактивен. Вы также можете позже экспериментировать с максимальной длиной сообщения (J1939_MSGLEN) и количеством поддерживаемых фильтров с сообщений (MSGFILTERS), но, пожалуйста, обратите внимание на сообщение компилятора Arduino. Это может предупредить вас о проблемах с объемом памяти.

Кроме того, не стесняйтесь просматривать файл, так как он позволяет вам задать дополнительных настраиваемых параметров, таких как:

- Предпочтительный исходный адрес
- Диапазон адресов
- НАЗВАНИЕ настроек

```
#define SA_PREFERRED 128
#define ADDRESSRANGEBOTTOM 129
#define ADDRESSRANGETOP 247

// Поля ИМЯ По
умолчанию 0xFFFFFFF
0xFFFF

0
0x00
0xFF 0
0x7F
0
0x00
```

#define NAME_ARBITRARY_ADDRESS_CAPABLE	0x01
--	------

Я повторяю създесъ (см. главу SAE J1939/81 — Заявление об адресе).

Процедура), но я считаю ссылку необходиимой на данный момент:

Примечание. Поля NAME были назначены таким образом, чтобы они не мешали при использовании в существующей сети транспортных средств. Это было сделано путем установки максимального значения идентификационного номера кода производителя, что приведет к более пастивной роли в процессе запроса адреса ECU с более высоким значением NAME с большей вероятностью проигрывает конкуренцию другому узлу, использующему тот же адрес.

Все показанные настройки используются только в демонстрационных целях. В результате вы должны следовать рекомендациям SAE. Кроме того, только вы (а не автор или издаватель) несете ответственность за окончательную реализацию и ее результаты.

3.2.2.2 Структура программы

Как и в любом приложении Arduino, инициализация данных и протокола J1939 происходит одновременно с функцией `setup()`, в то время как фактическое приложение находится в цикле `loop()`.

`setup()` — функция j1939.Init обязательна для работы ARD1939. Вам также необходимо установить предпочтительный адрес и ИМЯ (в целях тестирования вы можете использовать настройки проекта по умолчанию). Установка диапазона адресов не является обязательной.

ARD1939 не будет передавать данные (PGN), если вы не установите фильтр сообщений. ARD1939 поддерживает до 10 (Uno) или 100 (Mega 2560) фильтров сообщений.

`loop()` — эта функция должна начинаться с вызова `delay()`, за которым следует `j1939.Operate()`. Обязательно, чтобы время задержки времени, передаваемое в `j1939.Operate`, были идентичными, в противном случае синхронизация протокола будет отключена, что, в свою очередь, может привести к возникновению ошибок.

В следующем примере кода демонстрируется пример приложения с текаптактами J1939 с выделенными функциями, характерными для ARD1939:

```

{
    // Установить скорость передачи данных последовательного порта
    // интерфейса Serial.begin(115200);

    // Инициализировать протокол J1939, включая нас тройки
    CAN j1939.Init(SYSTEM_TIME);

    // Настройка фильтра сообщений
    PGN j1939.SetMessageFilter(PGN_ComponentID);

    // Установить предпочтительный адрес и диапазон
    // адресов j1939.SetPreferredAddress(128);
    j1939.SetAddressRange(128, 247);

    // Установить ИМЯ
    j1939.SetNAME(0x00, 0xFFFF, 0x00, 0x00, 0xFF, 0x7F, 0x00, 0x01, 0x00);

} // конец нас тройки

// недействительный цикл
0 {
    // Установить базу таймера в миллисекундах
    delay(SYSTEM_TIME);

    // Вызов с текстом протокола J1939
    nJ1939Status = j1939.Operate(&nMsgId, &IPGN, &pMsg[0], &nMsgLen, &nDestAddr,
        &nSrcAddr, &nPriority);

    // Проверяем получение PGN для нашего ECU/CA
    if(nMsgId == J1939_MSG_APP) {

        // Проверить статус протокола
        J1939 switch(nJ1939Status)
        {
            // Дело ADDRESSCLAIM_INPROGRESS; перерыв;

            case NORMALDATATRAFFIC: //
                Определяем соответствующий одиничный адрес
                nAppAddress = j1939.GetSourceAddress();

                // Ответ, соответствующий полученному PGN
                switch(IPGN) {

                    // Проверяем ваши PGN

                } // конец переключателя (IPGN)
                break;

            case ADDRESSCLAIM_FAILED: перерыв;
        }
    }
}
```

```
// конечный переключатель(j1939Status)

// конец , если

// конец цикла
```

Каждое примерное приложение Arduino, предоставленное в с ледующих главах , будет с ледовать этой же схеме.

Примечание. Эскиз ARD1939, который я предоставлю на странице загрузки, также соответствует этому шаблону . Однако я добавил пример кода, который я использовал для проверки концепции. Эти разделы кода примера закомментированы в основном файле, но вы, конечно, можете повторно активировать разделы, которые хотите использовать.

3.2.3 Подтверждение концепции

Естественно, с учетом того, что аппаратное обеспечение Arduino очень далеко от коммерчески доступных промышленных решений, некоторые сомневаю щиеся могут спорить даже тот факт, что программное обеспечение Arduino соответствует задаче запуска полного протоколов SAE J1939. Поэтому я считаю необъодимым выполнить и задокументировать базовое доказательство концепции. И даже если вы считаете такое доказательство излишним, взгляните на тестовые примеры, поскольку они показывают подробности взаимодействия узлов, которые интересны каждому программисту. Эти тесты объясняют фактический протокол SAE J1939 более подробно, чем любой другой учебник или руководство.

Примечание. Хотя я очень подробно описал характеристики протокола ARD1939, я не задокументировал все результаты тестов в этой книге, потому что не все эти сценарии предоставлены достаточную образовательную ценность.

Также важно отметить, что хотя моя тестовая среда записывает метки времени, значения времени, показанные на различных снимках экрана, отражают время, когда сообщения достигли фактической сети, т.е. они не обязательно отражают внутреннюю синхронизацию ECU.

По иронии судьбы, Arduino является идеальным аппаратным обеспечением для выполнения этих тестов, поскольку позволяет быстро настроить моделируемые неисправности. Следовательно, я использовал две системы Arduino, одну в качестве тестового объекта, а другую для имитации боевого протокола.

В частности, в качестве тестовой я использовал версию Arduino Mega 2560 (ARD1939-Mega).

объект, в то время как система Arduino Uno выс тупала в качестве сетевого симулятора. Чтобы проверить процессы утверждения адреса, а также процессы времени связи TP (Transport Protocol), я изменил код Arduino Uno, чтобы он выдавал ошибки связи времени. Тестовый узел (например, Mega 2560), в свою очередь, должен обнаруживать ошибки, создаваемые тестовым узлом.

Оба узла будут иметь начальный адрес 0x80 (128). Оба узла также используют один и те же настройки NAME, за исключением экземпляра ECU. Arduino Uno установил экземпляр ECU в свое имя NAME на «0», в то время как Arduino Mega использует экземпляр «1». Это гарантирует, что Uno всегда будет выигрывать в процессе подачи заявки на адрес против Mega.

Настройки в коде такие:

1. ARD1939.h

```
// поля имени по
умолчанию #define name_identity_number          0xFFFFFFF
#define name_manufacturer_code #define           0xFFFF
name_function_instance #define
name_ecu_instance #define name_funct           0x00 или
                                         0x01
                                         0xFF 0
                                         0x7F
                                         0
                                         0x00
                                         0x01
```

2. ARD1939 – настройка()

```
// Установить предпочтительный адрес и
диапазон адресов
j1939.SetPreferredAddress(SA_PREFERRED); j1939.SetAddressRange (ДИАПАЗОН АДРЕСОВ НИЖНИЙ, ДИАПАЗОН АДРЕСОВ ВЕРХНИЙ);

// Установить
Имя j1939.SetNAME(Имя_ИДЕНТИФИКАЦИОННЫЙ
                  НОМЕР, Имя_КОД_ИЗГотовите ля,
                  Имя_ФУНКЦИЯ_ЭКЗЕМПЛЯР,
                  Имя_ЭКУ_ИНСТАНЦИЯ,
                  Имя_ФУНКЦИЯ,
                  Имя_ТРАНСПОРТНОЕ
                  СРЕДСТВО_СИСТЕМА, Имя_ТРАНСПОРТНОЕ
                  СРЕДСТВО_СИСТЕМА_ЭКЗЕМПЛЯР,
                  Имя_ПРОМЫШЛЕННАЯ ГРУППА, Имя_АРБИТРАРИ_АДРЕС_КАПАБЕЛЬ);
```

Примечание. В соответствии с SAE J1939/81 каждый ECU (или лучшее управляемое приложение, поскольку ECU может поддерживать несколько СА) должен иметь 的独特的名称.

Для окончательного доказательства я ис пользовал с ним и экрана, сделанные с помощью программного обеспечения анализатора ADFweb CAN, чтобы задокументировать связь между двумя узлами, включая синхронизацию (т. е. измерение времени отклика итайм-аутов, как определено в стандартах SAE J1939).

Следующая информация также может служить общим для тестирования с стандарту SAE J1939. Однако нельзя утверждать, что описанные методы тестирования являются полными на 100 % или подходят для каждого сценария тестирования.

3.2.3.1 Процедура подачи заявки на адрес (SAE J1939/81)

Функционал протестированного устройства адресов включает в себя:

- Запрос адреса без конкурирующего узла
- Запрос адреса с конкурирующим узлом

3.2.3.1.1 Заявление адреса без конкурирующего узла

Первый и самый простой тест — проверить, существует ли узел (Arduino Mega 2560) в процессе сброса адреса. Этот тест выполняется путем просмотра включения и выключения питания (сброса) устройств, когда никакой другой конкурирующий узел (т. е. узел, конкурирующий за тот же адрес) не подключен.

NR	TIME	ID (HEX)	DATA (HEX)
1	1:11:07 AM.333.9	18EEFF80	FF FF FF FF 01 FF FE 80

На экране показано сообщение Address Claimed с идентификатором сообщения CAN 0x18EEFF80.

0x18 — приоритет сообщения = 6

0xEE — старший бит заявления адреса PGN (60928 = 0xEE00) 0xFF — младший бит заявления адреса PGN (адрес назначения), указывающий на широковещательное сообщение (использованием глобального адреса 255 в качестве адреса назначения)

0x80 — Зая вленный адрес (ис х одный адрес)

Для с равнения , давайте пос мотрим на тестовый узел (Arduino Uno), который запускается в тех же условиях .

NR	TIME	ID (HEX)	DATA (HEX)
1	1:21:04 AM.205.8	18EEFF80	FF FF FF FF 00 FF FE 80

Единственным отличием здесь является экземпляр ECU 0x00 по сравнению с 0x01 узла Arduino Mega (5 байт слева в разделе DATA).

3.2.3.1.2 Зая вление адреса с конкурирующим узлом

Теперь давайте установим два узла конкурировать за свой адрес, так как они оба используют один и тот же предпочтительный ис х одный адрес . Для этого важно учитывать, какой узел запускается первым, и результаты тестирования покажут разницу.

Тест №1: Сначала запускается Mega 2560, затем Uno

NR	TIME	ID (HEX)	DATA (HEX)
1	1:27:35 AM.123.7	18EEFF80	FF FF FF FF 01 FF FE 80
2	1:27:40 AM.445.7	18EEFF80	FF FF FF FF 00 FF FE 80
3	1:27:40 AM.446.8	18EEFF81	FF FF FF FF 01 FF FE 80

- Стока 1: Arduino Mega 2560 запускается и требует адрес 0x80 (128).
- Стока 2: Arduino Uno запускается и требует тот же адрес .
- Стока 3: Arduino Mega 2560 сравнил оба ИМЯ и обнаружил, что конкурирующий узел (Uno) имеет более низкое ИМЯ (из-за экземпляра ECU). Следовательно, Mega 2560 запрашивает следующий доступный адрес 0x81 (129).

узлов Примечание: Посмотрите на экземпляр ECU th байт данных) для идентификации двух (5 и их ответы.

Тест №2: Сначала запускается Uno, затем Mega 2560

NR	TIME	ID (HEX)	DATA (HEX)
1	1:37:00 AM.876.7	18EEFF80	FF FF FF FF 00 FF FE 80
2	1:37:07 AM.515.1	18EEFF80	FF FF FF FF 01 FF FE 80
3	1:37:07 AM.516.4	18EEFF80	FF FF FF FF 00 FF FE 80
4	1:37:07 AM.517.9	18EEFF81	FF FF FF FF 01 FF FE 80

Этот снимок экрана я с ним демонстрирует, что последовательность запуска приведет к другому процессу связи между двумя узлами.

- Стока 1: Arduino Uno запускается и требует адрес 0x80 (128).
- Стока 2: Arduino Mega 2560 запускается и требует тот же адрес.
- Стока 3: Arduino Uno проверил оба имени и обнаружил, что его имя ниже. Следовательно, он снова отправляет сообщение Address Claimed.
- Стока 4: Arduino Mega 2560 сравнил оба NAME и обнаружил, что конкурирующий узел (Uno) имеет более низкое NAME (из-за экземпляра ECU). Следовательно, Mega 2560 запрашивает следующий достаточно простой адрес 0x81 (129).

Следующее исключение становится еще интереснее. В данном случае я определил узел Arduino Mega 2560 как «несобственный к произвольному адресу», что означает, что узел будет иметь только один собственныйенный адрес.

Модификация кода:

1. ARD1939.h

```
// Поля ИМЯ по умолчанию
:
#define NAME_ARBITRARY_ADDRESS_CAPABLE 0x00
```

2. ARD1939 – настройка()

```
// Установить предпочтительный адрес и диапазон
адресов j1939.SetPreferredAddress(SA_PREFERRED); //
j1939.SetAddressRange(ADDRESSRANGEBOTTOM, ADDRESSRANGETOP);
```

```
// Установить
ИМЯ j1939.SetNAME(ИМЯ_ИДЕНТИФИКАЦИОННЫЙ
НОМЕР, ИМЯ_КОД_ИЗГOTОВИТЕЛЯ,
ИМЯ_ФУНКЦИЯ_ЭКЗЕМПЛЯР,
ИМЯ_ЭКУ_ИНСТАНЦИЯ,
ИМЯ_ФУНКЦИЯ,
ИМЯ_ТРАНСПОРТНОЕ
СРЕДСТВО_СИСТЕМА, ИМЯ_ТРАНСПОРТНОЕ
СРЕДСТВО_СИСТЕМА_ЭКЗЕМПЛЯР,
ИМЯ_ПРОМЫШЛЕННАЯ_ГРУППА, ИМЯ_АРБИТРАРИ_АДРЕС_КАПАБЕЛЬ);
```

Тест №3: Сначала запускается Mega 2560, затем Uno

NR	TIME	ID (HEX)	DATA (HEX)
1	1:58:11 AM.193.4	18EEFF80	FF FF FF FF 01 FF FE 00
2	1:58:16 AM.038.5	18EEFF80	FF FF FF FF 00 FF FE 80
3	1:58:16 AM.040.2	18EEFF80	FF FF FF FF 01 FF FE 00
4	1:58:16 AM.042.3	18EEFF81	FF FF FF FF 00 FF FE 80

Результат несколько удивителен (Mega выигрывает адресную претензию), но, в конце концов, он имеет смысл.

- Стока 1: Arduino Mega 2560 запускается и требует адрес 0x80 (128).
- Стока 2: Arduino Uno запускается и требует тот же адрес.
- Стока 3: Arduino Mega 2560 проверил оба NAME и обнаружил, что оба NAME ниже. Следовательно, он снова отправляет сообщение Address Claimed .
- Стока 4: Arduino Uno сравнил оба ИМЯ и обнаружил, что конкурирующий узел (Uno) имеет более низкое ИМЯ (не из-за экземпляра ECU, а из-за флага с возможностью произвольного адреса). Следовательно, Uno запрашивает следующий доступный адрес 0x81 (129).

Примечание: ИМЯ, как оно появляется в сообщении «Заданный адрес», начинается с младшего бита и постепенно старшего бита.

Тест №4: Сначала запускается Uno, затем Mega 2560

NR	TIME	ID (HEX)	DATA (HEX)
1	2:05:33 AM.304.7	18EEFF80	FF FF FF FF 00 FF FE 80
2	2:05:36 AM.691.5	18EEFF80	FF FF FF FF 01 FF FE 00
3	2:05:36 AM.693.0	18EEFF81	FF FF FF FF 00 FF FE 80

Результат очевиден, и я воздержусь здесь от объяснения всех подробностей.

Примечание. Этот последний тестовый сценарий скрывает одну из многих идиотских деталей протокола SAE J1939. Первоначально можно предположить, что ЭБУ с одним исходным адресом имеет некоторые недостатки по сравнению с ЭБУ с нескользкими доступными адресами. Однако положение флагка «С возможностью произвольного адреса» гарантирует, что узлы, «менее способные», чем другие, все же могут подключаться к сети.

На следующем этапе мы создадим сценарии, в которых узел не может запросить исходный адрес. Для этой цели мы установим для обоих узлов флаг «Возможность произвольного адреса» в «0» и будем использовать один и тот же адрес 0x80 (128).

Тест №5: Сначала запускается Mega 2560, затем Uno

NR	TIME	ID (HEX)	DATA (HEX)
1	12:11:23 AM.901.2	18EEFF80	FF FF FF FF 01 FF FE 00
2	12:11:23 AM.902.4	18EEFF80	FF FF FF FF 00 FF FE 00
3	12:11:23 AM.904.0	18EEFFFE	FF FF FF FF 01 FF FE 00

Это первый сценарий, когда один из двух узлов, именно Arduino Mega 2560, сети не может запросить исходный адрес.

- Строка 1: Arduino Mega 2560 запускается и требует исходный адрес 0x80 (128).
- Строка 2: Arduino Uno запускается и требует тот же адрес.
- Строка 3: Arduino Mega 2560 сравнивает оба NAME и определяет, что конкурирующий узел имеет более высокий приоритет (более низкое NAME). Поскольку у узла нет других доступных адресов, он отправляет сообщение Cannot Claim Address .

Примечание. Сообщение «Невозможно запросить адрес» идентично сообщению «Адрес занят», но в нем указывается ПУСТОЙ адрес (254) в качестве исходного адреса.

Тест №6: Сначала запускается Uno, затем Mega 2560

NR	TIME	ID (HEX)	DATA (HEX)
1	1:00:09 AM.358.2	18EEFF80	FF FF FF FF 00 FF FE 00
2	1:00:10 AM.889.8	18EEFF80	FF FF FF FF 01 FF FE 00
3	1:00:10 AM.890.9	18EEFF80	FF FF FF FF 00 FF FE 00
4	1:00:10 AM.892.6	18EEFFFE	FF FF FF FF 01 FF FE 00

Здесь снова (как и ожидалось) Arduino Uno побеждает в процессе запроса адреса.

- Стока 1: Arduino Uno запускается и требует исходный адрес 0x80 (128).
- Стока 2: Arduino Mega 2560 запускается и требует тот же адрес.
- Стока 3: Arduino сравнил оба NAME и обнаружил, что его NAME имеет более высокий приоритет (более низкий NAME). Как следствие, он снова отправляет сообщение Address Claimed.
- Стока 4: Arduino Mega 2560 сравнивает оба NAME и определяет, что конкурирующий узел имеет более высокий приоритет (более низкое NAME). Поскольку у узла нет других доступных адресов, он отправляет сообщение Cannot Claim Address.

И последнее, но не менее важное: здесь идет самый интересный тест, потому что он включает в себя серьезное моделирование сети. Вопрос в том, правильно ли наш ECU J1939 управляет доступным диапазоном адресов?

С этой целью мы добавляем код в прикладную программу Arduino Uno, который позволит нам отклонить определенный набор адресов, занятых другими Arduino Mega 2560.

Условия испытаний:

- Оба ECU имеют предпочтительный исходный адрес 0x80 (128).
- Оба ECU поддерживают произвольные адреса.
- Оба ECU поддерживают сегментированный диапазон адресов от 129 до 135.
- Arduino Uno имеет экземпляр ECU 0x00.
- Arduino Mega 2560 имеет экземпляр ECU 0x01.

В обоих скетчах, Uno и Mega 256, мы меняем и добавляем следующий код

в функции `setup()` (с м. выделенные разделы):

```
// Установить предпочтительный адрес и диапазон
адресов j1939.SetPreferredAddress(SA_PREFERRED);
j1939.SetAddressRange(129, 135);
```

Дополнение к функции `loop()` Arduino Uno показано здесь:

```
:
:

байт msgFakeNAME[] = {0, 0, 0, 0, 0, 0, 0, 0};

// Устанавливаем базу таймера в миллисекундах
задержка(SYSTEM_TIME);

// Вызов с тела протокола J1939
nJ1939Status = j1939.Operate(&nMsgId, &IPGN, &pMsg[0], &nMsgLen, &nDestAddr, &nSrcAddr, &nPriority);

// Блокировать определенные заявленные
адреса if(nMsgId == J1939_MSG_PROTOCOL) { if(IPGN
== 0x00EE00) { if(nSrcAddr >=
129 && nSrcAddr <= 135) j1939.Transmit(6, 0x00EE00,
nSrcAddr, 255, msgFakeNAME, 8);

// конец, если

} // конец, если

// Проверяется получение PGN для нашего ECU/CA
if(nMsgId == J1939_MSG_APP)
:
:
```

Во-первых, мы используем «фальшивое» ИМЯ, чтобы гарантировать, что Uno выигрывает процесс с претензии, несмотря ни на что. Имя, заполненное всеми нулями, делает именно это. В конце концов, это программист.

Задачей `j1939.Operate(...)` мы можем определить, является ли полученный PGN сообщением приложения (`nMsgId = J1939_MSG_APP`) или сообщением управления сетью (`nMsgId = J1939_MSG_PROTOCOL`).

Если PGN является сообщением управления сетью, мы проверяем, является ли это сообщением `Address Claimed` или нет. Далее проверяем диапазон адресов, заявленный

другой узел.

Затем следует «неприятная» строка, в которой мы отказываем Arduino Mega 2560 в любом адресе, который он пытается получить, вызывая функцию j1939Transmit(...). Мы передаем сообщение Address Claimed и используем тот же исходный адрес, который заявлен в Mega 2560. Поскольку NAME Uno неприведено, Mega 2560 будет пытаться требовать больше адресов, пока не закончатся ресурсы.

На следующем скриншоте экрана показано именно то, что я описал:

NR	TIME	ID (HEX)	DATA (HEX)
1	1:54:55 AM.940.1	18EEFF80	FF FF FF FF 01 FF FE 80
2	1:54:55 AM.941.8	18EEFF80	FF FF FF FF 00 FF FE 80
3	1:54:55 AM.944.0	18EEFF81	FF FF FF FF 01 FF FE 80
4	1:54:55 AM.945.5	18EEFF81	00 00 00 00 00 00 00 00
5	1:54:55 AM.946.8	18EEFF82	FF FF FF FF 01 FF FE 80
6	1:54:55 AM.948.2	18EEFF82	00 00 00 00 00 00 00 00
7	1:54:55 AM.949.6	18EEFF83	FF FF FF FF 01 FF FE 80
8	1:54:55 AM.950.9	18EEFF83	00 00 00 00 00 00 00 00
9	1:54:55 AM.952.3	18EEFF84	FF FF FF FF 01 FF FE 80
10	1:54:55 AM.953.6	18EEFF84	00 00 00 00 00 00 00 00
11	1:54:55 AM.955.1	18EEFF85	FF FF FF FF 01 FF FE 80
12	1:54:55 AM.956.2	18EEFF85	00 00 00 00 00 00 00 00
13	1:54:55 AM.957.9	18EEFF86	FF FF FF FF 01 FF FE 80
14	1:54:55 AM.960.0	18EEFF86	00 00 00 00 00 00 00 00
15	1:54:55 AM.961.7	18EEFF87	FF FF FF FF 01 FF FE 80
16	1:54:55 AM.963.8	18EEFF87	00 00 00 00 00 00 00 00
17	1:54:55 AM.965.6	18EEFFFE	FF FF FF FF 01 FF FE 80

Сначала Arduino Mega 2560 пытается заявить свой предпочтительный адрес 0x80 (128), а после отказывается заявить права со 129 (0x81) по 135 (0x87).

Наконец (с строкой 17), у него заканчиваются адреса, и он отправляет сообщение Cannot Claim Address.

В качестве финального теста мы бросим Mega 2560 кость и разрешим адрес 135 (0x87). Это означает, что в функции loop() Uno меняет один параметр, т.е. устанавливаем последний адрес на 134 вместо 135.

```
// Блокировать определенные
заявленные адреса if(nMsgId ==
J1939_MSG_PROTOCOL {
```

```

if(IPGN == 0x00EE00) { if(nSrcAddr
>= 129 && nSrcAddr <= 134) j1939.Transmit(6, 0x00EE00,
nSrcAddr, 255, msgFakeNAME, 8);

}// конец , если

}// конец , если

```

NR	TIME	ID (HEX)	DATA (HEX)
1	2:22:41 AM.080.2	18EEFF80	FF FF FF FF 01 FF FE 80
2	2:22:41 AM.081.8	18EEFF80	FF FF FF FF 00 FF FE 80
3	2:22:41 AM.083.0	18EEFF81	FF FF FF FF 01 FF FE 80
4	2:22:41 AM.084.5	18EEFF81	00 00 00 00 00 00 00 00
5	2:22:41 AM.085.7	18EEFF82	FF FF FF FF 01 FF FE 80
6	2:22:41 AM.087.2	18EEFF82	00 00 00 00 00 00 00 00
7	2:22:41 AM.088.5	18EEFF83	FF FF FF FF 01 FF FE 80
8	2:22:41 AM.089.8	18EEFF83	00 00 00 00 00 00 00 00
9	2:22:41 AM.091.3	18EEFF84	FF FF FF FF 01 FF FE 80
10	2:22:41 AM.092.5	18EEFF84	00 00 00 00 00 00 00 00
11	2:22:41 AM.094.1	18EEFF85	FF FF FF FF 01 FF FE 80
12	2:22:41 AM.095.2	18EEFF85	00 00 00 00 00 00 00 00
13	2:22:41 AM.096.8	18EEFF86	FF FF FF FF 01 FF FE 80
14	2:22:41 AM.098.9	18EEFF86	00 00 00 00 00 00 00 00
15	2:22:41 AM.100.7	18EEFF87	FF FF FF FF 01 FF FE 80

Как видно из скриншота, Mega 2560 перебирает все доступные адреса, пока не доходит до 135 (0x87). Наконец, он может запросить адрес (с строка 15).

3.2.3.2 Отправка и получение с сообщений

Доказательство обнос ти с амой простой задачи отправки и получения обычных 8-байтовых (CAN) с сообщений с ледует расматривать как не нужный тест, поскольку это уже имелось в предыдущих главах.

Однако все меняется при передаче и приеме кадров данных J1939, с содержащих более восемьми байтов.

Обратите внимание, что для широковещательной передачи с сообщений и RTS/CTS для одноранговой связи, основаны на обмене несколькими пакетами данных и требуют точного контроля времени. Другими словами, оба функции протокола имеют рядтайм-аутов, которые необходимо обладать.

Примечание. Стандарт SAE J1939/21 требует, чтобы ECU одновременно поддерживал один сеанс ВАМ и один сеанс RTS/CTS, и с теком протоколов ARD1939 выполняет это требование.

3.2.3.3 Транс портный протокол — сеанс ВАМ (SAE J1939/21)

Тестовые элементы включают в себя:

- Основная функция
- Контроль времени

Для начального теста сеанса ВАМ требуется только один ECU, в этом случае я использую полный протокол, работающий на Arduino Mega 2560 (Arduino Uno все еще подключен к сети, поэтому Mega 2560 потребует всего один адрес 0x81 = 129).

Код, который мы добавили в функцию loop() Mega 2560, выделен:

```
:
:
// Переменные для проверки
концеpt или byte msgLong[] = {0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
                                0x40, 0x41, 0x42, 0x43, 0x44, 0x45};

// Установить базу таймера в миллисекундах
delay(SYSTEM_TIME);

// Вызов текущего протокола J1939
nJ1939Status = j1939.Operate(&nMsgId, &IPGN, &pMsg[0], &nMsgLen, &nDestAddr,
                             &nSrcAddr, &nPriority);

// Отправляем периодическое сообщение длиной более 8 байт
// Сеанс ВАМ
if(nJ1939Status == NORMALDATATRAFFIC) { nCounter+

+;

if(nCounter == (int)(5000/SYSTEM_TIME))

{ nSrcAddr = j1939.GetSourceAddress();
j1939.Передача(6, 59999, nSrcAddr, 255, msgLong, 15);
nСчетчик = 0;

} // конец , если

} // конец , если
```

:

:

Во-первых, мы используем глобальную переменную nCounter, для которой установлено значение по умолчанию, равное нулю.

Внутри функции loop() мы созаем «длинное» сообщение с 15 байтами данных. Затем мы проверяем статус запроса адреса и продолжаем, когда будет достигнут «нормальный трафик данных», что означает, что процесс запроса адреса прошел успешно.

Каждые пять секунд (выбирается с лучшим образом) мы определяем полученный из одногого адреса и отправляем PGN (59999, также выбранный с лучшим образом) с использованием длинного сообщения. Мы также используем адрес назначения 255, глобальный адрес, который должен инициализировать с вами.

Давайте посмотрим на скриншот экрана, сделанный с помощью программы инструмента анализа ADFweb CAN:

NR	TIME	ID (HEX)	DATA (HEX)
1	12:13:56 AM.942.4	1CECFF81	20 0F 00 03 FF 5F EA 00
2	12:13:56 AM.996.9	1CEBFF81	01 31 32 33 34 35 36 37
3	12:13:57 AM.051.5	1CEBFF81	02 38 39 40 41 42 43 44
4	12:13:57 AM.106.1	1CEBFF81	03 45 FF FF FF FF FF FF

Линия 1:

- ЭБУ отправляет вам (с сообщение о широковещательном сообщении).
 - ID (HEX): 0x1C
 - указывает приоритет 7. 0xECFF
 - иницирует TP (транспортный протокол), что означает, что это TP.CM (управление с единением) PGN из 0xEC00 (60416) плюс глобальный адрес 0xFF (255). 0x81 (129) — исходный адрес ЭБУ.
 -
 - ДАННЫЕ (HEX): 0x20
 - = 32 = управляющий байт, указывающий на вам 0x000F
 - указывает общую длину 15 байтов данных (с начала младший бит, последний MSB) 0x03
 - указывает количество пакетов для передачи 0xFF (зарезервировано,
 - согласно SAE J1939/21) 0x00EA5F указывает PGN (59999)
 - мультипакетного сообщения.

(с начала младший бит, пос ледний с старший бит)

Строка 2-с строка 4:

- Данные, отправленные ECU по адресу 0x81 (129)
- Первый байт указывает порядковый номер, в данном случае от 1 до 3. Следующие 7 байтов
- представляют собой необработанные данные.
- Все неиспользуемые данные в последнем пакете устанавливаются на 0xFF

3.2.3.3.1 Синхронизация с сообщений

NR	TIME	ID (HEX)	DATA (HEX)
1	12:13:56 AM.942.4	1CECFF81	20 0F 00 03 FF 5F EA 00
2	12:13:56 AM.996.9	1CEBFF81	01 31 32 33 34 35 36 37
3	12:13:57 AM.051.5	1CEBFF81	02 38 39 40 41 42 43 44
4	12:13:57 AM.106.1	1CEBFF81	03 45 FF FF FF FF FF

Обратите внимание на метки времени от строки 1 до строки 4 снимка экрана (посмотрите на последние четыре числа, обозначающие время в десятых миллисекундах; например, 942,4 миллисекунды в строке 1). SAE J1939/21 требует частоты пакетов от 50 до 200 миллисекунд, что соответствует ЭБУ Mega 2560.

Тайм-аут возникает, когда между двумя пакетами с сообщениями прошло более 750 миллисекунд, хотя ожидалось больше пакетов. Тайм-аут закроет соединение, но не будет обмена с сообщениями, указывающими на обнаруженный тайм-аут.

Примечание. В этом предыдущем примере платы Arduino Uno играли лишь незначительную роль в сети J1939. Как следствие, с установленным протоколом ARD1939-Uno/TP он получит сообщение ВАМ. Однако для того, чтобы сообщение достигло фактического приложения, вам необходимо установить фильтр, чтобы разрешить передачу PGN в приложение.

3.2.3.4 Транспортный протокол — сейс RTS/CTS (SAE J1939/21)

Тестовые элементы включают в себя:

- Основная функция

- Контроль времени

Естественно, для сеанса RTS/CTS нам понадобятся два узла SAE J1939. В моей настройке это означает связь между Arduino Uno и ЭБУ Mega 2560.

Условия тестирования аналогичны предыдущим примерам:

- Стек протоколов SAE J1939 установлен в обеих системах.
- Arduino Uno имеет предпочтительный адрес 0x80 (128) и экземпляр ECU «0».
- Arduino Mega имеет предпочтительный адрес 0x80 (128) и экземпляр ECU «1», что означает, что после процесса аутверждения адреса Mega 2560 получит 0x81 (129) в качестве исходного адреса.
- Arduino Mega отправит 15-байтовое сообщение (с копированное из предыдущего примера) на Arduino Uno.

Изменения кода в функции loop() Mega 2560 минимальны:

```
:
:
// Переменные для проверки
концепции byte msgLong[] = {0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38,
    0x39, 0x40, 0x41, 0x42, 0x43, 0x44, 0x45};

// Установить базу таймера в миллисекундах
delay(SYSTEM_TIME);

// Вызов сеанса протокола J1939
nJ1939Status = j1939.Operate(&nMsgId, &IPGN, &pMsg[0], &nMsgLen, &nDestAddr,
    &nSrcAddr, &nPriority);

// Отправляя ем периодическое сообщение длиной более 8 байт
// Сеанс RTS/CTS
if(nJ1939Status == NORMALDATATRAFFIC)

{
    nCounter++;

    if(nCounter == (int)(5000/SYSTEM_TIME))

    {
        nSrcAddr = j1939.GetSourceAddress();
        j1939.Transmit(6, 59999, nSrcAddr, 0x80, msgLong, 15);
        nСчетчик = 0;
    }
}

// конец, если
```

```
// конец, если
:
:
```

Вместо отправки сообщения на глобальный адрес (255) мы отправим его на адрес Arduino Uno 0x80.

Нам также нужно изменить код в функции `setup()` Arduino Uno, т.е. нам нужно разрешить передачу PGN, установив фильтр с сообщений. Вставляя емый (выделенный) код выглядит следующим образом:

```
// Установить предпочтительный адрес и диапазон
адресов j1939.SetPreferredAddress(SA_PREFERRED);
j1939.SetAddressRange (диапазон адресов нижний, диапазон адресов верхний);

// Установить фильтр
с сообщений j1939.SetMessageFilter(59999);

// Установить имя
j1939.SetNAME(Имя_идентификационный номер,
    Имя_код_изголовите ля,
    Имя_функция_экзэмпляр,
    Имя_эку_инстанция,
    Имя_функция,
    Имя_трансポートное
    СРЕДСТВО_СИСТЕМА, Имя_трансポートное
    СРЕДСТВО_СИСТЕМА_экзэмпляр,
    Имя_промышленная группа, Имя_арбитра_адрес_капабель);
```

Если мы не установим фильтр с сообщений, мы увидим следующее с сообщение нашине:

NR	TIME	ID (HEX)	DATA (HEX)
1	2:18:19 AM.814.8	1CEC8081	10 0F 00 03 FF 5F EA 00
2	2:18:19 AM.816.8	1CEC8180	FF 02 FF FF FF 5F EA 00

- Стока 1: Mega 2560 передает сообщение «Запрос на отправку» (адрес источника = 0x80, адрес назначения 0x81).
- Стока 2: Arduino Uno отвечает сообщением о прекращении соединения (обратите внимание на первый байт данных = 255) и заявляет, что ему не хватает необходимых ресурсов (см. второй байт данных = 0x02).

Примечание. Стандарт SAE J1939/21 не расматривает все возможные сценарии, которые

может привести к сообщению «Connection Abort». Ясно необходиимо предотвратить обратную связь, когда сеанс RTS/CTS не может быть продолжен. В этом случае причина в том, что узлу не нужно сообщение.

Все становится понятным интересным, когда Arduino Uno устанавливает фильтр, позволяющий ПИН:

NR	TIME	ID (HEX)	DATA (HEX)
1	2:43:00 AM.295.2	1CEC8081	10 0F 00 03 FF 5F EA 00
2	2:43:00 AM.296.3	1CEC8180	11 03 01 FF FF 5F EA 00
3	2:43:00 AM.352.4	1CEB8081	01 31 32 33 34 35 36 37
4	2:43:00 AM.407.1	1CEB8081	02 38 39 40 41 42 43 44
5	2:43:00 AM.461.8	1CEB8081	03 45 FF FF FF FF FF FF
6	2:43:00 AM.464.0	1CEC8180	13 0F 00 03 FF 5F EA 00

Линия 1:

- ECU по адресу 0x81 (129) отправляет сообщение Request to Send на ECU по адресу 0x80 (128).
- ID (HEX): 0x1C
- указывает приоритет 7. 0xEC80
- инициирует TP (транспортный протокол), что означает, что это TP.CM (управление соединением) PGN из 0xEC00 (60416) плюс адрес назначения 0x80 (128). 0x81 (129) — это одинаковый адрес ЭБУ.
-
- ДАННЫЕ (HEX): 0x10
- = 16 = управляющий байт, указывающий сообщение запроса на отправку 0x000F
- указывает общую длину 15 байтов данных (с начала младший бит, последний старший бит) 0x03 указывает количество пакетов для передачи 0xFF
- (зарезервировано в соответствии с SAE J1939 /21)
- 0x00EA5F указывает PGN (59999) мультипакетного сообщения.

(с начала младший бит, последний старший бит)

Строка 2:

- ЭБУ по адресу 0x80 (128) отправляет сообщение «Готово к отправке» на ЭБУ по адресу 0x81 (129)
- Идентификатор (шестнадцатеричный):

- 0x1C указывает приоритет 7. 0xEC81
- инициализирует TP (транс портный протокол), что означает, что это PGN TP.CM (управление с единением) 0xEC00 (60416) плюс адрес назначения 0x81 (129). 0x80 (128) — исходный адрес ЭБУ.
-
- **ДАННЫЕ (HEX):** 0x11 =
17 = управляющий байт, указывающий на сообщение «Готово к отправке» 0x03
указывает количество пакетов для передачи 0x01 указывает номер следующего пакета
Следующие два байта зарезервированы (0xFF) 0x00EA5F
указывает PGN (59999) мультипакетного сообщения
(с начала младший бит, последний старший бит)

Строка с 3 по строку 5: данные,

- отправленные ЭБУ по адресу 0x81 (129) в ЭБУ по адресу 0x80 (128).
- Первый байт указывает порядковый номер, в данном случае от 1 до 3. Следующие 7 байтов представляют собой необработанные данные.
- Все неиспользуемые данные в последнем пакете устанавливаются на 0xFF

Строка 6:

- ECU по адресу 0x80 (128) отправляет сообщение End of Message
Подтверждение сообщение для ЭБУ по адресу 0x81 (129)
- ID (HEX): 0x1C
- указывает приоритет 7. 0xEC81
- инициализирует TP (транс портный протокол), что означает, что это TP.CM (управление с единением) PGN из 0xEC00 (60416) плюс адрес назначения 0x81 (129). 0x80 (128) — исходный адрес ЭБУ.
-
- **ДАННЫЕ (HEX):** 0x13 =
19 = управляющий байт, указывающий конец сообщения
Сообщение подтверждения 0x000F
- указывает длину сообщения (с начала младший бит, последний старший бит) 0x03
- указывает общее количество пакетов
- Следующий байт зарезервирован (0xFF)

- 0x00EA5F указывает PGN (59999) мультипакетного сообщения.

(с начала младший бит, пос ледний с старший бит)

3.2.3.4.1 Синхронизация с сообщениями

NR	TIME	ID (HEX)	DATA (HEX)
1	2:43:00 AM.295.2	1CEC8081	10 0F 00 03 FF 5F EA 00
2	2:43:00 AM.296.3	1CEC8180	11 03 01 FF FF 5F EA 00
3	2:43:00 AM.352.4	1CEB8081	01 31 32 33 34 35 36 37
4	2:43:00 AM.407.1	1CEB8081	02 38 39 40 41 42 43 44
5	2:43:00 AM.461.8	1CEB8081	03 45 FF FF FF FF FF FF
6	2:43:00 AM.464.0	1CEC8180	13 0F 00 03 FF 5F EA 00

Обратите внимание на метки времени от строк 3 до строки 5 снимка экрана (посмотрите на последние четыре числа, обозначающие время в десятых миллисекундах; например, 352,4 миллисекунды в строке 3). SAE J1939/21 требует частоты пакетов от 50 до 200 миллисекунд, что соответствует ЭБУ Mega 2560.

3.2.3.4.2 Таймаут Clear to Send

При запуске теста таймаута Clear to Send выполняется при запуске Mega 2560 ECU в качестве единственного узла сети.

Чтобы проверить этот сценарий, я запускаю Mega 2560 по адресу 0x80 (128) и пытаюсь синхронизировать сеанс RTS/CTS с узлом по адресу 0x33 (51).

NR	TIME	ID (HEX)	DATA (HEX)
1	1:56:31 AM.050.7	18EEFF80	FF FF FF FF 01 FF FE 80
2	1:56:36 AM.743.8	1CEC3380	10 0F 00 03 FF 5F EA 00
3	1:56:36 AM.961.6	1CEC3380	FF 03 FF FF FF 5F EA 00

- Стока 1: Arduino Mega 2560 требует адрес 0x80.
- Стока 2: отправляет сообщение «Запрос на отправку» узлу по адресу 0x33.
- Стока 3: спустя немногим более 200 миллисекунд передается сообщение Connection Abort, в котором в качестве причины прерывания указывается таймаут.

3.2.3.4.3 Таймаут подтверждения окончания сообщения

Чтобы протестировать все таймауты во время сеанса RTS/CTS, я ис пользовал специальный скетч Arduino для имитации всех «неприятных» функций, которые могут вызвать сбой связи. Этот метод было проще реализовать, чем ис пользовать полный стандарт SAE J1939. Работа скодом с текстом протоколов для имитации усложнила ошибки и оказалось трудоемкой и не привела к улучшению реального кода (на самом деле, с одним делом, с одним наоборот, поскольку это увеличило бы размер кода).

Как и во всех предыдущих примерах, этот скетч Arduino (реализованный на Uno) довольно просто предоставляет вам все возможности для имитации неисправного узла J1939.

Во-первых, нам нужно определить сообщение Clear to Send в глобальной памяти:

- байт msg_CTS[] = {0x11, 0x03, 0x01, 0xFF, 0xFF, 0x5F, 0xEA, 0x00};

Данные основаны на ранее описанном сеансе RTS/CTS (см. раздел ДАННЫЕ в строке 2).

NR	TIME	ID (HEX)	DATA (HEX)
1	2:43:00 AM.295.2	1CEC8081	10 0F 00 03 FF 5F EA 00
2	2:43:00 AM.296.3	1CEC8180	11 03 01 FF FF 5F EA 00
3	2:43:00 AM.352.4	1CEB8081	01 31 32 33 34 35 36 37
4	2:43:00 AM.407.1	1CEB8081	02 38 39 40 41 42 43 44
5	2:43:00 AM.461.8	1CEB8081	03 45 FF FF FF FF FF FF
6	2:43:00 AM.464.0	1CEC8180	13 0F 00 03 FF 5F EA 00

Скетч Arduino, т. е. функция loop(), был адаптирован для этой конкретной сессии RTS/CTS:

```
недействительный цикл
{
// Байт
объявления nPriority
= 0; длинный
IPGN = 0; байт
nSrcAddr = 0; байт
nDestAddr = 0;
байт nData[8]; интервал ндатаген;

// Проверка полученных сообщений J1939
if(j1939Receive(&IPGN, &nPriority, &nSrcAddr, &nDestAddr, nData, &nDataLen)
```

```

== 0)

{ switch(IPGN) { case

0xEC00: // Сеанс RTS/CTS PGN

// Проверяется переключатель

формата CTS(nData[0])

{ case 0x10: // Запрос на отправку

// Отправить Очистить для

отправки j1939Transmit(0xEC00, 7, 0x33, 0x80, &msg_CTS[0], 8); Serial.print("Запрос на

отправку.\n\r"); перерыв;

case 0xFF: // Прерывание соединения

Serial.print("Соединение прервано.\n\r"); перерыв;

}// конечный переключатель

перерыв;

помолчанию:

перерыв;

}// конечный переключатель

}// конец , если

}// конец цикла

```

Как только функция j1939Receive(...) возвращает допустимый фрейм данных, мы проверяем PGN. Если PGN является PGN транс портного протокола (TP), мы смотрим на первый байт в масиве данных, чтобы определить, является ли сообщение запросом на отправку или сообщением о разрыве соединения.

Если это запрос на отправку, Arduino передает кадр данных Clear to Send. Однако с кетч не отправит подтверждение окончания сообщения, и это должно вызвать сообщение об отмене соединения от Mega 2560.

NR	TIME	ID (HEX)	DATA (HEX)
1	2:23:58 AM.081.7	1CEC3380	10 0F 00 03 FF 5F EA 00
2	2:23:58 AM.082.8	1CEC8033	11 03 01 FF FF 5F EA 00
3	2:23:58 AM.138.8	1CEB3380	01 31 32 33 34 35 36 37
4	2:23:58 AM.193.5	1CEB3380	02 38 39 40 41 42 43 44
5	2:23:58 AM.248.2	1CEB3380	03 45 FF FF FF FF FF FF
6	2:23:59 AM.608.5	1CEC3380	FF 03 FF FF FF 5F EA 00

- Стока 1: Mega 2560 отправляет сообщение Request to Send .
- Стока 2: Uno передает сообщение Clear to Send .
- Стока 3-5 : Mega передает данные.
- Стока 6: после таймаута без получения подтверждения окончания с сообщения Mega отправляет сообщение о прекращении соединения с последовательной линией на таймаут.

3.2.3.4.4 Программирование сеанса RTS/CTS

Ради интереса я расширил ранее используя скетч Arduino, чтобы имитировать полнофункциональный сеанс RTS/CTS (один же, код сильно адаптирован к предыдущему сеансу RTS/CTS). На основе этого скетча могли моделировать все случаи ошибок на стороне получателя .

NR	TIME	ID (HEX)	DATA (HEX)
1	3:13:52 AM.814.9	1CEC3380	10 0F 00 03 FF 5F EA 00
2	3:13:52 AM.815.9	1CEC8033	11 03 01 FF FF 5F EA 00
3	3:13:52 AM.872.0	1CEB3380	01 31 32 33 34 35 36 37
4	3:13:52 AM.926.7	1CEB3380	02 38 39 40 41 42 43 44
5	3:13:52 AM.981.4	1CEB3380	03 45 FF FF FF FF FF FF
6	3:13:52 AM.982.7	1CEC8033	13 0F 00 03 FF 5F EA 00

На этом снимке экрана показан полный сеанс RTS/CTS с скетчем Arduino Uno, высвечивающим в качестве моделируемого партнера по связи.

Без объяснения каждой отдельной строки (что было сделано ранее), полная сессия RTS/CTS прошла в соответствии с SAE J1939/81.

Модифицированный код выглядит следующим образом (с м. выделенные разделы):

Объем глобальной памяти:

```
байт msg_CTS[] = {0x11, 0x03, 0x01, 0xFF, 0xFF, 0x5F, 0xEA, 0x00}; байт msg_ACK[] =
{0x13, 0x0F, 0x00, 0x03, 0xFF, 0x5F, 0xEA, 0x00}; интервал nPackages = 0; интервал
nCounter = 0;
```

Функция Цикла():

```
недействительный цикл
{
    // Байт
    объявлены nPriority
    = 0; длинный
    IPGN = 0; байт nSrcAddr
    = 0; байт nDestAddr = 0;
    байт nData[8];
    интервал nDataLen;

    // Проверка полученных сообщений J1939
    if(j1939Receive(&IPGN, &nPriority, &nSrcAddr, &nDestAddr, nData, &nDataLen)
    == 0)

    { switch(IPGN)

        { case 0xEC00: // Сеанс RTS/CTS PGN

            // Проверяет переключатель
            формата CTS(nData[0])

            { case 0x10: // Запрос на отправку

                // Отправить Очистить
                для отправки j1939Transmit(0xEC00, 7, 0x33, 0x80, &msg_CTS[0], 8);
                Serial.print("Запрос на отправку.\n\r");

                // Запоминаем количество пакетов nPackages
                = (int)nData[3];

                перерыв;

                case 0xFF: // Прерывание соединения
                Serial.print("Соединение прервано.\n\r");
                перерыв;

            } // конечный переключатель

            перерыв;

        case 0xEB00: // Пакет данных RTS/CTS

            nСчетчик++;
            Serial.print("Пакет: ");

    }
```

```

Serial.print(nCounter); Serial.print
("из"); Serial.print(nPackages);
Serial.print("\n\r");

если (nCounter == nPackages) {

    // Прибыл последний пакет данных
    j1939Transmit(0xEC00, 7, 0x33, 0x80, &msg_ACK[0], 8);

    Serial.print("Отправка ACK.\n\r"); nСчетчик = 0;

}

// конец , если

помолчанию:

перерыв;

// конечный переключатель

}

// конец , если

}

// конец цикла

```

Примечание. Как видите, этот программный проект доступен на странице загрузки по адресу <http://ard1939.com>.

3.2.4 ARD1939 — пример приложения

Типичное приложение ECU SAE J1939 включает в себя не только стек протоколов, но и большой объем обработки ввода и/или вывода. Это может включать считывание датчиков и отправку результата в виде PGN или считывание PGN и настройку выхода (цифрового или аналогового).

Однако, поскольку мое внимание сосредоточено на стеке протоколов J1939, а не на программировании ввода-вывода Arduino, я не буду вдаваться в подробности до тела к аппаратным интерфейсам Arduino. Существует множество хороших книг, которые более чем достаточно освещают эту конкретную тему (см. также мою рекомендацию в приложении по рекомендованной литературе).

Тем не менее, я включил в проект ARD1939 краткий пример для обработки запросов:

- PGN 65242 — Идентификация программного обеспечения

- PGN 65259 — Идентификация компонентов
- PGN 65267 — Положение автомобиля

Во-первых, давайте посмотрим на эти PGN:

- PGN 65242 — Идентификация программного обеспечения
 - Скорость передачи: по запросу
 - Длина данных: переменная
 - Приоритет по умолчанию: 6
 - Номер PG: 65242 (FEDAhex)
 - Данные:
 - Байт 1: количество полей идентификатора программного обеспечения (зависит от пользователя)
 - Байты 2...n: идентификация программного обеспечения
- PGN 65259 — Идентификация компонентов
 - Скорость передачи: по запросу
 - Длина данных: переменная
 - Приоритет по умолчанию: 6
 - Номер PG: 65259 (FEEBhex)
 - Данные:
 - Байт 1–5: Сделать
 - Переменная, до 200 символов: Модель
 - Переменная, до 200 символов: Серийный номер
 - Переменная, до 200 символов: Номер услуги

Обобщение, программное обеспечение и идентификатор компонента могут содержать не сколько (не обязательно) полей, как описано, которые должны быть разделены знаком «*» (разделитель). Не обязательно включать все поля, однако разделитель «*» обязательно включен всегда.

- PGN 65267 — Положение автомобиля
 - Скорость передачи: 5 сек.
 - Длина данных: 8 байт
 - Приоритет по умолчанию: 6
 - Номер PG: 65267 (FEF3hex)
 - Данные:
 - Байт 1–4: Широта
 - Байт 5–8: Долгота

Что касается программирования этих PGN в нашем приложении, нам сначала нужно составить с одержимое с сообщения (в данном случае только для демонстрационных целей) и определить PGN.

В глобальном пространстве памяти (выше `setup()`) мы добавляем следующее:

```
// PGN и сообщения
#define PGN_RequestMessage 0xEA00

#define PGN_SoftwareID 65242 #define
PGN_ComponentID 65259 #define
PGN_VehiclePosition 65267

#define MSGLEN_SoftwareID 12 #define
MSGLEN_ComponentID 33 #define
MSGLEN_VehiclePosition 8

byte msgSoftwareID[] = {"V*1*00*01**"}; byte
msgComponentID[] = {"MAKEX*MODEL_123*01234567890-007**"}; байт
msgVehiclePosition[] = {0x08, 0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01};
```

Примечание. Содержание всех сообщений было выбрано с лучшим образом.

В подпрограмме `setup()` нам нужно определить различные фильтры с сообщений, в данном случае только для сообщения запроса (идентификатор программы обес печения, идентификатор компонента и положение транс порта не принимаются; это ответы от нашего приложения J1939).

```
// Установить фильтр
с сообщений j1939.SetMessageFilter(PGN_RequestMessage);
```

Мы ставим все остальные настройки, такие как предпочитаемый исходный адрес, диапазон адресов и имя, со значениями по умолчанию (см. файл ARD1939.h).

Важно знать, что существует обе функции ARD1939, связанные с обработкой сообщения запроса:

Как определено в стандарте SAE J1939, сообщение запроса имеет вид 0xEA00, где младший бит используется в качестве адреса получателя, т. е. адрес ЭБУ, который должен предоставить запрошенную информацию (например, 0xEA80, где 0x80 — адрес получателя).

Однако фильтр с сообщениями с текущими протоколами ARD1939 позволяет

с сообщение в диапазоне 0xEAxx, которое будет передаваться при установке любого фильтра PGN в диапазоне 0xEAxx.

Такое поведение необходимо, чтобы разрешить глобальный адрес (255), что означает, что существуют случаи, когда один ECU запрашивает информацию со всех узлов сети.

Как следствие, обязательно, чтобы приложение, в дополнение к сообщению запроса, также сбрасывало адрес назначения со своим собственным адресом.

В функцию `loop()` мы добавляем код для проверки сообщения запроса и запрошенног о PGN, как показано ниже:

```
// Вызов стека протокола J1939
nJ1939Status = j1939.Operate(&nMsgId, &IPGN, &pMsg[0], &nMsgLen, &nDestAddr, &nSrcAddr,
                             &nPriority);

// Проверяется получение PGN для нашего ECU/CA
if(nMsgId == J1939_MSG_APP) {

    // Проверить статус протокола
    J1939 switch(nJ1939Status)

    { case ADDRESSCLAIM_INPROGRESS:

        перерыв;

        case НОРМАЛЬНЫЙ ТРАФИК ДАННЫХ :

            // Определяем байт со старшим адресом
            nAppAddress; nAppAddress = j1939.GetSourceAddress();

            // Ответ, соответствующий полученному
            PGN

            switch(IPGN) { case PGN_RequestMessage:

                // Проверяем, что мы являемся поставщиком запрошенной
                информацией if(nDestAddr == GLOBALADDRESS
                           || nDestAddr == nAppAddress) {

                    // Определяем запрошенный PGN; обратите внимание, что LSB
                    // идет первым long lRequestedPGN = ((long)pMsg[2]
                    // << 16) + ((long)pMsg[1] <<
                    // 8) + (long)pMsg[0];

                    // ПРИМЕЧАНИЕ. Далее nSrcAddr — это адрес
```

```

запрашиваю щий узел.

switch(lRequestedPGN) { case

    PGN_SoftwareID: Serial.print("Отправка
        идентификатора прог раммног о обес печения.\n\r");

        // Добавля ем количест во полей в с сообщение SW ID msgSoftwareID[0] = 4;

        j1939.Transmit(6, PGN_SoftwareID, nAppAddress, nSrcAddr, &msgSoftwareID[0], MSGLEN_SoftwareID);

        перерыв;

    case PGN_ComponentID: Serial.print("Отправка
        идентификатора компонента.\n\r"); j1939.Transmit(6, PGN_ComponentID, nAppAddress,
        nSrcAddr, &msgComponentID[0], MSGLEN_ComponentID);

        перерыв;

    case PGN_VehiclePosition: Serial.print("Отправка
        положения автомобиля.\n\r"); j1939.Transmit(6, PGN_VehiclePosition,
        nAppAddress, nSrcAddr, &msgVehiclePosition[0], MSGLEN_VehiclePosition);

        перерыв;

    } // конечный переключатель

} // конец , если

перерыв;

} // конечный переключатель(IPGN)

перерыв;

случай ADDRESSCLAIM_FAILED:

перерыв;

} // конечный переключатель(nJ1939Status)

} // конец , если

```

Примечание. Добавленный код для этого приложения J1939 будет работать на обеих аппаратных версиях Arduino, Uno и Mega 2560.

Чтобы протестировать приложение, на этот раз я не использовал другой Arduino, а ввел данные для сообщения запроса вручную в программный инструмент ADFweb, как показано здесь:

NR	ID(HEX)	DESCRIPTION	DATA (HEX)
1	18EA8033	Request for Software ID	DAFE00
2	18EA8033	Request for Component ID	EBFE00
3	18EA8033	Request for Vehicle Position	F3FE00

- Стока 1: Узел 0x33 запрашивает идентификацию программного обеспечения от узла 0x80.
- Стока 2: Узел 0x33 запрашивает Идентификацию Компонента от узла 0x80.
- Стока 3: Узел 0x33 запрашивает позицию транспортного средства от узла 0x80.

Примечание. Поскольку Arduino является единственным узлом J1939 в сети, он будет запрашивать адрес 0x80 (128). В программном инструменте ADFweb я использовал 0x33 в качестве адреса запрашиваемой стороны.

Распечатка на последовательном мониторе Arduino выглядит так, как и ожидалось:



Однако трафик данных J1939, записанный на шину, показало сначала неожиданность, но, естественно, потом все это имело смысл.

NR	TIME	ID (HEX)	DATA (HEX)
1	12:18:29 AM.851.0	18EEFF80	FF FF FF FF 01 FF FE 80
2	12:18:48 AM.757.0	18EA8033	DA FE 00
3	12:18:48 AM.759.7	1CEC3380	10 0C 00 02 FF DA FE 00
4	12:18:48 AM.976.0	1CEC3380	FF 03 FF FF FF DA FE 00
5	12:18:53 AM.274.0	18EA8033	EB FE 00
6	12:18:53 AM.276.9	1CEC3380	10 21 00 05 FF EB FE 00
7	12:18:53 AM.493.2	1CEC3380	FF 03 FF FF FF EB FE 00
8	12:18:59 AM.220.6	18EA8033	F3 FE 00
9	12:18:59 AM.222.8	18FEF380	08 07 06 05 04 03 02 01

- Стока 1: Arduino требует адрес 0x80.

- Стока 2: Сообщение запроса на идентификацию программного обеспечения.
- Стока 3: Arduino отправляет сообщение Request to Send .
- Стока 4: Arduino отправляет сообщение о прерывании ссоединения из-за таймаута.

- Стока 5: Сообщение запроса идентификации компонента.
- Стока 6: Arduino отправляет сообщение «Запрос на отправку» .
- Стока 7: Arduino отправляет сообщение о прерывании ссоединения из-за таймаута.

- Стока 8: Запрос с сообщения о местоположении автомобиля .
- Стока 9: Arduino отправляет данные о положении автомобиля .

Во-первых , в строках 8 и 9 нет ничего удивительного. PGN для местоположения автомобиля (65267 = 0xEF03) не попадает в диапазон адресов для одноранговой связи, поэтому идентификатор сообщения содержит только адрес отправителя (0x80), то есть адрес Arduino.

Однако в случае запросов с сообщений для идентификации программного обеспечения и компонентов ARD1939 сообщает о таймауте перед тем как сообщения о прекращении соединения . Оба сообщения , программное обеспечение и идентификатор компонента , имеют длину более восьми байтов и , следовательно , требуют использования транспортного протокола (TP) , в данном случае сенсора RTS/CTS , поскольку это одноранговая связь . Таймаут вызван тем , что в сети нет узла J1939 с адресом 0x33 (а инструмент ADFweb действует только как пассивный член сети).

Чтобы установить связь без ошибок таймаута , нам нужно подключить другой узел Arduino с адресом источника 0x33 , а в подпрограмме setup() нам нужно установить фильтры для всех трех PGN , идентификацию программного обеспечения (65242) , идентификацию компонентов (65259) и положение автомобиля (65267) .

Наконец , результат , как и ожидалось :

NR	TIME	ID (HEX)	DATA (HEX)
1	12:22:47 AM.491.9	18EEFF33	FF FF FF FF 00 FF FE 80
2	12:24:22 AM.154.6	18EEFF80	FF FF FF FF 01 FF FE 80
3	12:24:44 AM.399.3	18EA8033	DA FE 00
4	12:24:44 AM.402.4	1CEC3380	10 0C 00 02 FF DA FE 00
5	12:24:44 AM.404.5	1CEC8033	11 02 01 FF FF DA FE 00
6	12:24:44 AM.460.2	1CEB3380	01 04 56 2A 31 2A 30 30
7	12:24:44 AM.514.5	1CEB3380	02 2A 30 31 2A 2A FF FF
8	12:24:44 AM.516.0	1CEC8033	13 0C 00 02 FF DA FE 00
9	12:24:55 AM.763.2	18EA8033	EB FE 00
10	12:24:55 AM.766.3	1CEC3380	10 21 00 05 FF EB FE 00
11	12:24:55 AM.768.2	1CEC8033	11 05 01 FF FF EB FE 00
12	12:24:55 AM.824.1	1CEB3380	01 4D 41 4B 45 58 2A 4D
13	12:24:55 AM.878.4	1CEB3380	02 4F 44 45 4C 5F 31 32
14	12:24:55 AM.932.7	1CEB3380	03 33 2A 30 31 32 33 34
15	12:24:55 AM.987.0	1CEB3380	04 35 36 37 38 39 30 2D
16	12:24:56 AM.041.3	1CEB3380	05 30 30 37 2A 2A FF FF
17	12:24:56 AM.042.8	1CEC8033	13 21 00 05 FF EB FE 00
18	12:25:04 AM.589.0	18EA8033	F3 FE 00
19	12:25:04 AM.591.2	18FEF380	08 07 06 05 04 03 02 01

- Стока 1: Один Arduino требует адрес 0x33.
- Стока 2: адрес друг их утверждений Arduino 0x80.
- Стока 3: Сообщение запроса на идентификацию программы раммног о обес печения .
- Стока 4-8 : Идентификация программы раммног о обес печения передается и подтверждается .
- Стока 9: Сообщение запроса идентификации компонента.
- Стока 10-17: Идентификация компонента передается и подтверждается .
- Стока 18: Запрос с сообщения о местоположении автомобиля .
- Стока 19: передается положение автомобиля .

4. Вывод

Со всеми предыдущими примерами программирования, установленными и обьясненными, вы теперь должны хорошо разбираться в программировании с собственным приложением SAE J1939 с помощью Arduino Uno и/или Mega 2560.

Тем не менее, все еще существует множество возможных сценариев для приложений J1939, и все они могут иметь свои собственные проблемы. Тем не менее, эта книга с ее практическим подходом должна помочь вам избежать крутой кривой обучения. Я считаю, что прилагаемые снимки экрана с трафиком данных SAE J1939 очень помогают понять протокол J1939.

Как я упоминал ранее, основное внимание в этой книге уделялось реализации полнофункционального синтаксиса протоколов SAE J1939 с использованием аппаратного обеспечения Arduino. Я не рассматривал стандартные возможности ввода/вывода Arduino и многие дополнительные приложения для считывания данных с датчиков, которые приходят на ум. Такие приложения, безусловно, потребуют дополнительного оборудования, в частности щитов Arduino, и все они будут сопровождаться длительным исследованием их возможностей.

На данный момент я оставлю поиск более сложных приложений на вас, но подумаю о написании еще одной книги по этой теме.

Всего доброго!

Приложение А. Макросы отладки

Имейте в виду, что следующие макросы являются односимвольными. Из-за ограниченного места на странице некоторые макросы отображаются как заключенные в более чем одну строку, а разрывы строк обозначаются символом «\».

```
#define DEBUG_INIT() char sDebug[128];

#define DEBUG_PRINTHEX(T, v) {Serial.print(T); sprintf(sDebug, "%x\n\r", v); \
Serial.print(sDebug);}

#define DEBUG_PRINTDEC(T, v) {Serial.print(T); sprintf(sDebug, "%d\n\r", v); \
Serial.print(sDebug);}

#define DEBUG_PRINTARRAYHEX(T, a, l) {Serial.print(T); if(l == 0) \
{Serial.print("Пусто.\n\r");} else {for(int x=0; x<l; x++){sprintf(sDebug, "%x", a[x]); \
Serial.print(sDebug);} Serial.print("\n\r");}}
```

```
#define DEBUG_PRINTARRAYDEC(T, a, l) {Serial.print(T); if(l == 0) \
Serial.print("Пусто.\n\r"); else {for(int x=0; x<l; x++){sprintf(sDebug, "%d", a[x]); \
Serial.print(sDebug);} Serial.print("\n\r");}}
```

```
#define DEBUG_HALT() { пока(Serial.available() == 0); Serial.setTimeout(1); \
Serial.readBytes(sDebug, 1);}
```

Приложение В. Стек протоколов ARD1939 Ссылка

Функции, доступные для прикладного уровня SAE J1939 (т.е. вашей программы)

Являются:

Инициализация

- j1939.Init – Инициализирует нас тройки стека протоколов
- j1939.SetPreferredAddress – Задает адрес предпочтительного узла (источника)
- j1939.SetAddressRange – Задает диапазон возможных адресов (не обязательно)
- j1939.SetNAME – Задает ИМЯ ЭБУ с использованием индивидуальных параметров PGN для обработки
- вашем приложении

Чтение/запись — проверка состояния

- j1939.Operate — обрабатывает процесс утверждения адреса, считывает PGN из сети автомобиля и передает текущий статус протокола (выполняется утверждение адреса, успешное утверждение адреса, неудавшееся утверждение адреса) j1939.Transmit —
- передает данные в сеть автомобиля и обрабатывает Транспортный протокол (TP)

Другие функции приложения

- j1939.Terminate — сбрасывает нас тройки стека протоколов.
- j1939.GetSourceAddress — предоставляет собственный адрес узла.
- j1939.DeleteMessageFilter — удаляет фильтр сообщений.

Структура приложения

Как и в любом приложении Arduino, инициализация данных и протокола J1939 происходит одновременно с функцией `setup()`, в то время как фактическое приложение находится в цикле `loop()`.

`setup()` — функция j1939.`Init` обрабатывается для работы ARD1939. Вам также необходимо установить предпочтительный адрес и имя (в целях тестирования вы можете использовать настройки проекта по умолчанию). Установка диапазона адресов не является обязательной.

ARD1939 не будет передавать данные (PGN), если вы не установите фильтр сообщений. ARD1939 поддерживает до 10 (UNO) или 100 (Mega 2560) фильтров сообщений.

`loop()` — эта функция должна начинаться с вызова `delay()`, за которым следует `j1939.Operate()`. Обязательно, чтобы время задержки времени, передаваемое в `j1939.Operate`, были идентичными, в противном случае с их разницей протокола будет отключен, что, в свою очередь, вызовет состояния ошибки.

Настройки функциональности находятся внутри файла `ARD1939.h`:

```
// Версия
// программы // -----
// 0 - ARD1939-Uno
// 1 - ARD1939-Uno/TP
// 2 - ARD1939-Mega
#define ARD1939VERSION
```

Вызовы функций Описание

j1939.Init

недействительным j1939.Init (int nSystemTime);

Инициализирует память ARD1939, настройки и корости передачи и т. д.

nSystemTime — это время цикла вашего приложения в миллисекундах. Эта информация предоставит текущим протоколам ARD1939 базу времени для управления всеми таймерами, необходимыми для различных задач протокола.

В идеале системное время должно составлять 1 миллисекунду для лучшей производительности, однако до 10 миллисекунд должно быть достаточно для обычного сетевого трафика. Можно использовать любые более высокие значения, но это может поставить под угрозу производительность.

j1939.SetPreferredAddress

недействительным j1939.SetPreferredAddress (байт nAddr);

Устанавливает предпочтительный адрес узла. Значение по умолчанию — 128 (см. файл ARD1939.h). Этот вызов функции является обязательным для инициализации текущих протоколов; в противном случае не может отправляться сообщения в сеть автомобиля. Предпочтительный адрес не зависит от диапазона сегментованных адресов, т. е. он может быть установлен в любом месте внутри или за пределами этого диапазона.

nAddr — это предпочтительный адрес узла (источника). Он должен находиться в диапазоне от 128 до 252. Адреса ниже 128 разрешены, но регулируются стандартом SAE J1939.

j1939.SetAddressRange

недействительным j1939.SetAddressRange (байт nAddrBottom, байт nAddrTop);

Устанавливает сегментированный диапазон адресов. Диапазон по умолчанию (для файла ARD1939.h) составляет от 129 до 247. Этот вызов функции является необязательным, что означает, что текущие протоколы будут работать только с предпочтительным адресом.

nAddrBottom определяет нижнюю часть диапазона сегментированных адресов.

nAddrTop определяет вершину диапазона сегментированных адресов.

```
j1939.SetNAME void  
j1939.SetNAME(long lIdentityNumber, int nManufacturerCode, байт  
nFunctionInstance, байт nECUInstance, байт nFunction, байт nVehicleSystem,  
байт nVehicleSystemInstance, байт nIndustryGroup,  
байт nArbitraryAddressCapable);
```

Устанавливает НАЗВАНИЕ ЭБУ с ис пользованием индивидуальных параметров.

Ниже показаны настройки по умолчанию для ИМЯ устроства, найденные в файле ARD1939.h :

#define name_identity_number #define	0xFFFFFFF
name_manufacturer_code #define	0xFFFF
name_function_instance #define	
name_ecu_instance #define	0x00
name_funct	

nArbitraryAddressCapable должен быть установлен равным нулю , если ваше приложение не поддерживает согласованный диапазон адресов (см. функцию j1939SetAddressRange).

Поля NAME были назначены таким образом, что они не будут мешать при ис пользовании в существующей сети транспортных средств. Это было сделано путем установки максимального значения идентификатора и кода производителя , что приведет к более пастивной роли в процессе запроса адреса ECU с более высоким значением NAME с большей вероятностью проигрывает конкуренту другому узлу, ис пользуя шестнадцатеричный формат .

Примечание. Все показанные настройки ис пользуются только в демонстрационных целях . В результате вы должны следовать рекомендациям SAE. Кроме того, только вы (а не автор или издаватель) несете ответственность за окончательную реализацию и ее результаты.

j1939.SetMessageFilter
байт j1939.SetMessageFilter(long IPGN);

Установливает PGN для обработки в вашем приложении. ARD1939 поддерживает до 10 (UNO) или 100 (Mega 2560) фильтров с сообщений.

IPGN — это PGN, который вы разрешаете передавать вашему приложению.

Функция возвращает OK или ERROR (как определено в ARD1939.h), где ERROR означает, что больше нет доступных фильтров с сообщений.

Обычный случай — с сообщение запроса:

Как определено в стандарте SAE J1939, с сообщение запроса имеет вид 0xEA00, где младший бит используется в качестве адреса получателя, т. е. адрес ЭБУ, который должен предсказать запрошенную информацию (например, 0xEA80, где 0x80 — адрес получателя).

Однако фильтр с сообщений с текущими протоколами ARD1939 позволит пропустить каждое сообщение в диапазоне 0xEAxx, если вы установите любой фильтр PGN в диапазоне 0xEAxx.

Такое поведение необходимо, чтобы разрешить глобальный адрес (255), что означает, что существует сценарии, когда один ECU запрашивает информацию со всех узлов сети.

Как следствие, обязательно, чтобы приложение, в дополнение к сообщению запроса, также содержит адрес назначения со своим собственным адресом.

J1939.Operate
byte j1939.Operate(byte* nMsgId, long* IPGN, byte* pMsg, int* nMsgLen,
byte* nDestAddr, byte* nSrcAddr, byte* nPriority);

Обрабатывает процесс запроса адреса, считывает PGN из сети автомобиля и собщает текущий статус протокола (выполняется запрос адреса, успешный запрос адреса, отказ запроса адреса).

Функция возвращает ADDRESSCLAIM_INPROGRESS, заявка успешна),
NORMALDATATRAFFIC (Адрес или
ADDRESSCLAIM_FAILED.

Параметры, передаваемые функции, являются указателями на:

nMsgId = J1939_MSG_NONE — сообщение не получено или J1939_MSG_APP — сообщение получено.

IPGN = PGN полученного сообщения

pMsg = Mac сив данных сообщения

nMsgLen = Размер масива данных сообщения

nDestAddr = адрес получателя сообщения (обычно адрес источника вашего приложения, но также может быть и глобальный адрес 255 — широковещательная строка с сообщений)

nSrcAddr = адрес источника, т. е. адрес узла, отправившего сообщение.

nPriority = приоритет сообщения.

j1939.Передать байт

j1939.Передать(байт nPriority, длинный IPGN, байт nSourceAddress, байт nDestAddress, байт* pData, int nDataLen);

Передает данные в автомобильную сеть и обрабатывает транспортный протокол (TP), то есть обрабатывает сообщения данных длиной от 0 до 1785 байт (для Arduino Uno это число ограничено 256). Функция автоматически вызывает транспортный протокол (TP), если длина сообщения превышает 8 байт.

В функцию передаются следующие параметры:

IPGN = PGN сообщения

pMsg = Mac сив данных сообщения

nMsgLen = Размер масива данных с сообщения

nDestAddr = адрес назначения с сообщения (также может быть глобальным адресом 255 для широковещательной передачи с сообщения)

nSrcAddr = Адрес источника, т.е. адрес вашего ECU.

nPriority = приоритет с сообщения .

j1939.Terminate void

j1939.Terminate(void);

Сбрасывает нас тройки с текущими протоколами.

j1939.GetSourceAddress байт

j1939.GetSourceAddress(void);

Представляет сброшенный адрес узла; будет NULLADDRESS (254) в случае ошибки процесса аутентификации адреса.

j1939.DeleteMessageFilter

недействительным j1939.DeleteMessageFilter (длинный IPGN);

Удаляет фильтр с сообщений.

IPGN = PGN, подлежащий удалению . Любые попытки удалить ранее установленный PGN будут игнорироваться .

Приложение С – Рекомендуемая литература

Существует более чем много полезной литературы по Arduino, но, будучи опытным программистом, я прочитал единственную работу:

Программирование Ардуино
Начало работы с эскизами
Саймон Монк
ISBN 978-0071784221

Также рекомендуется для представления дополнительной справочной информации:

Прототипирование локальной сети контроллеров (CAN) с помощью Arduino
Уилфрид Вос с
ISBN 978-1938581168

Полное руководство по локальной сети контроллеров
Уилфрид Вос с
ISBN 978-0976511601

Полный путеводитель по J1939
Уилфрид Вос с
ISBN 978-0976511632

Все книги Уилфрида Восса можно найти в книжных интернет-магазинах, таких как Amazon и Amazon Kindle (по всему миру), Apple Bookstore, Barnes & Noble, вкл. NOOK, Lulu.com (загружавшиеся в формате PDF), Kobo, Abebooks.com (включая все международные веб-сайты) и любой другой хороший книжный магазин.