Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.03 Прикладная информатика**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине «Микропроцессорные системы»
на тему:

Адаптер автомобильных диагностических шин

Студент ИУ6-74Б (Группа) _____ (Подпись, дата) М.А. Маркин (И.О. Фамилия)

Руководитель _____ (Подпись, дата) С.В. Ибрагимов (И.О. Фамилия)

2022 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский институт)»

УТВЕРЖДАЮ
Заведующий кафедрой ИУ6
_____/_____/
«___» _____ 2022 г.

ЗАДАНИЕ

на выполнение курсовой работы

по дисциплине «Микропроцессорные системы»

Студент Маркин М.А. (ИУ6-74Б)
(фамилия, инициалы, индекс группы)

Направленность курсовой работы – учебная
Источник тематики – кафедра
График выполнения работы: 25% – 4 нед., 50% – 8 нед., 75% – 12 нед., 100% – 16 нед.

Тема курсовой работы: Адаптер автомобильных диагностических шин

Разработать на основе микроконтроллера семейства STM32 устройство, обеспечивающее возможность передачи данных между ПЭВМ и информационными шинами автомобиля согласно стандарту SAE J2534. Поддерживать работу с протоколом CAN, ISO 15765-4. Обеспечить взаимодействие устройства с ПЭВМ посредством интерфейса USB.

Разработать схему, алгоритмы и драйверы устройства. Отладить разработанную программу и проверить ее работу на макетной плате.

Оценить и измерить потребляемую мощность устройства.

Оформление курсовой работы

1. Расчетно-пояснительная записка на 30 листах формата А4.
2. Перечень графического материала курсовой работы:
   а) функциональная электрическая схема;
   б) принципиальная электрическая схема.

Дата выдачи задания «___» _____ 2022 г.
Руководитель курсовой работы _____/_____/
Задание получил _____/_____/
«___» _____ 2022 г.

Примечание. Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

РЕФЕРАТ

Расчетно-пояснительная записка 37 с., 19 рис., 2 табл., 20 источн., 5 прил.

CAN-ШИНА, STM32, SAE J2534, ПРОЕКТИРОВАНИЕ УСТРОЙСТВА, ИНТЕРФЕЙС

Данная работа описывает процесс разработки устройства.

Объектом разработки является устройство, соответствующее стандарту SAE J2534, реализующее интерфейс CAN.

Цель работы – спроектировать конечное устройство, способное взаимодействовать с ПЭВМ согласно стандарту SAE J2534 и реализующее взаимодействие с протоколами CAN и ISO 15765.

В ходе разработки были решены следующие задачи:

- разработана структурная схема устройства,
- разработана функциональная схема устройства,
- разработана принципиальная схема устройства,
- спроектирована печатная плата устройства,
- разработано и отлажено ПО устройства.

В результате было разработано устройство, удовлетворяющее требованиям технического задания. Устройство соответствует требованиям стандарта SAE J2534 и позволяет ПЭВМ осуществлять взаимодействие с автомобильной шиной CAN.

Конечное устройство предназначено как для личного, так и для коммерческого использования для диагностики автомобилей.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

CAN – Controller Area Network,

COM – communication port,

DC – постоянный ток,

DoCAN – Diagnostics over CAN,

ISO – Международная организация по стандартизации,

LDO – линейный стабилизатор,

SAE – общество автомобильных инженеров,

SWD – serial wire debug,

USB – Universal Serial Bus,

USB-IF – USB Implementers Forum,

VCP – virtual COM-port,

АЦП – аналого-цифровой преобразователь,

БС – бортовая сеть,

ИС – интегральная схема,

МК – микроконтроллер,

ПО – программное обеспечение,

ПЭВМ – персональная ЭВМ,

САПР – система автоматизированного проектирования,

ТЗ – техническое задание.

ВВЕДЕНИЕ

Данная работа описывает разработку устройства-адаптера для автомобильных диагностических шин. Описаны этапы проектирования устройства и разработки программного обеспечения микроконтроллера.

В качестве основного вычислительного устройства выбран микроконтроллер семейства STM32 – STM32F105RB. Выбранный МК имеет аппаратные контроллер интерфейса USB 2.0 FS и два контроллера CAN 2.0B, необходимые для выполнения поставленной задачи.

В процессе выполнения работы был проведен анализ технического задания и необходимых международных стандартов. Были разработаны функциональная и принципиальная схемы устройства, печатная плата устройства, ПО микроконтроллера. Было выполнено отладочное тестирование устройства с использованием макета.

Разработка устройства, согласно техническому заданию, состоит из двух основных частей: конструкторская и технологическая часть.

Конструкторская часть включает в себя:

- разработка электрической функциональной схемы устройства;
- разработка электрической принципиальной схемы устройства;
- проектирование печатной платы устройства;
- разработка программного обеспечения устройства.

Технологическая часть включает в себя:

- описание программных и аппаратных инструментов, использованных при разработке устройства;
- описание методов тестирования и отладки устройства.

1 Конструкторская часть

1.1 Анализ технического задания

Техническое задание описывает устройство-адаптер, способное обеспечить взаимодействие ПЭВМ с автомобильными диагностическими шинами. Протокол взаимодействия необходимо реализовать в соответствии со стандартом SAE J2534 [1]. Устройство должно поддерживать работу со следующими протоколами: ISO 11898 (CAN) [2], ISO 15765 (DoCAN) [3].

Устройство подразумевается для использования в легковых автомобилях, следовательно устройство должно иметь возможность питаться от бортовой сети автомобиля.

Интерфейсом взаимодействия с ПЭВМ, согласно заданию, должен выступать протокол USB.

В результате анализа технического задания получаем, что устройство должно осуществлять передачу данных между ПЭВМ, используя интерфейс USB, и автомобильными шинами CAN.

Для взаимодействия с шиной CAN на физическом уровне необходимы CAN-трансивер. Для работы на канальном уровне необходим CAN-контроллер. Данные устройства могут быть объединены и представлять единое устройство, либо часть этих устройств может присутствовать в МК.

Для взаимодействия с ПЭВМ посредством протокола USB также необходим внешний или внутренний USB-контроллер.

1.2 Обзор возможных решений

Для выбора варианта реализации устройства сведем возможные варианты в таблицу и оценим их целесообразность с интерфейсной, производственной и экономической точек зрения в расчете на один канал CAN.

В таблице 1 представлены результаты сравнения возможных решений, где «интерфейс» – способ взаимодействия МК с CAN-контроллером, «кол-во устройств» – количество необходимых микросхем помимо МК, «стоимость» – стоимость данных устройств на момент разработки.

Таблица 1 – Сравнение возможных структурных решений

Вариант	МК и внешний CAN контроллер со встроенным трансивером	МК и внешние CAN-контроллер и CAN-трансивер	МК со встроенным CAN-контроллером и CAN-трансивер
Интерфейс	SPI	SPI	Встроенный
Кол-во устройств	1	2	1
Стоимость	Высокая	Средняя	Средняя

По совокупности параметров преимущество имеет вариант 3 – микроконтроллер со встроенным CAN-контроллером и внешним CAN-трансивером. Он имеет меньшее количество устройств по сравнению с вариантом 2, а значит проще в производстве, при сопоставимой стоимости компонентов. Интерфейс взаимодействия с CAN-контроллером в данном варианте, по сути, отсутствует, так как контроллер CAN встроен в МК, что обеспечивает большую интеграцию и упрощает работу разработчика.

1.3 Выбор компонентов устройства

1.3.1 Выбор микроконтроллера

Исходя из решения принятого в 1.2 нам необходим микроконтроллер, имеющий в своем составе два CAN-контроллера и способный работать с протоколом USB. Скорости его работы должно быть достаточно, чтобы обслуживать работу двух независимых шин CAN, работающих на скорости до 500 кбод/с.

По данным критериям из семейства STM32 подходит несколько серий микроконтроллеров:

– микроконтроллеры серии F1, G4 из линейки Mainstream;

– микроконтроллеры серии F2, F4, F7, H7 из линейки High-Performance;

– микроконтроллеры серии L4 из линейки Ultra-Low-Power.

На момент выполнения работы самым доступным вариантом являются МК серии F1 из линейки Mainstream. По итогу был выбран МК STM32F105RB, как самый доступный по стоимости за единицу и имеющийся на складах вариант.

Характеристики STM32F105RB [4]:

– ядро ARM Cortex-M3 32-бит до 72 МГц;
– ОЗУ 64 Кбайт SRAM;
– Flash-память 128 КБ;
– USB 2.0 OTG FS с 1,25 КБ выделенной памяти, on-chip PHY;
– 2x CAN 2.0B с 512 Б выделенной памяти;
– 2x 12-битных ЦАП
– 7 таймеров: 4 16-битных, 1 PWM, 2 DAC таймера, SysTick, 2 WDT;
– 5 USART,
– 3 SPI до 18 Мбит/с.
– напряжение питания от 2 В до 3,6 В.

Выбранный МК производится в различных корпусах, для разрабатываемого устройства был выбран вариант в корпусе LQFP-64. Конфигурация выводов показана на рисунке 1.

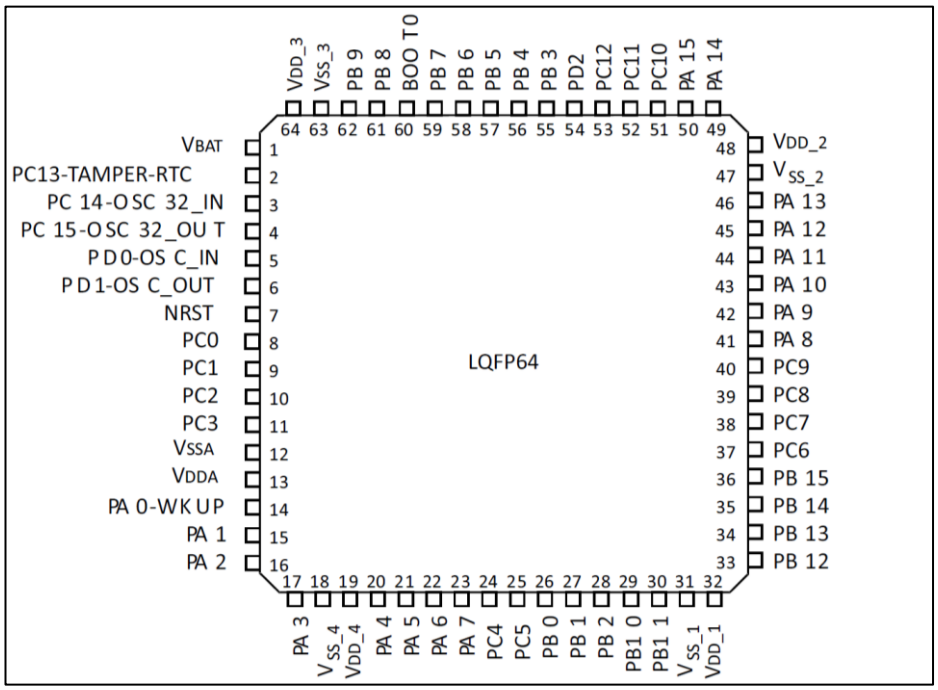

Рисунок 1 – Конфигурация выводов STM32F105RB

Общая для микропроцессоров F105 и F107 структурная диаграмма представлена на рисунке 2. Основное отличие данных МК – это наличие у F107 интерфейса Ethernet и отсутствие второго I^2C.

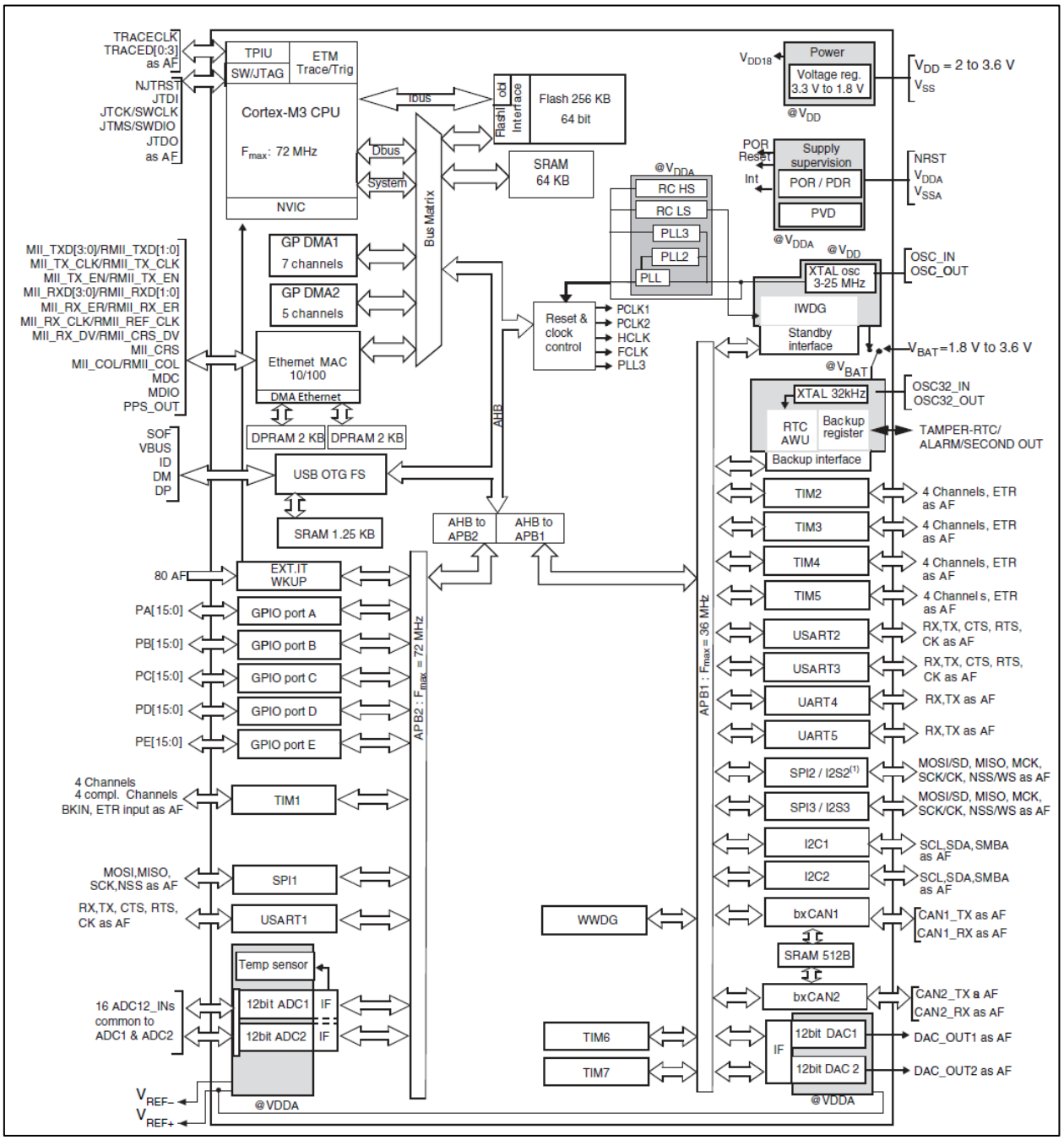

Рисунок 2 – Структурная диаграмма микроконтроллера

Микроконтроллер обладает 32-битной шиной адреса, что позволяет ему адресовать до 4 ГБ данных. Адресное пространство поделено между различными внутренними устройствами. Структура распределения адресного пространства памяти микроконтроллера представлена на рисунке 3.

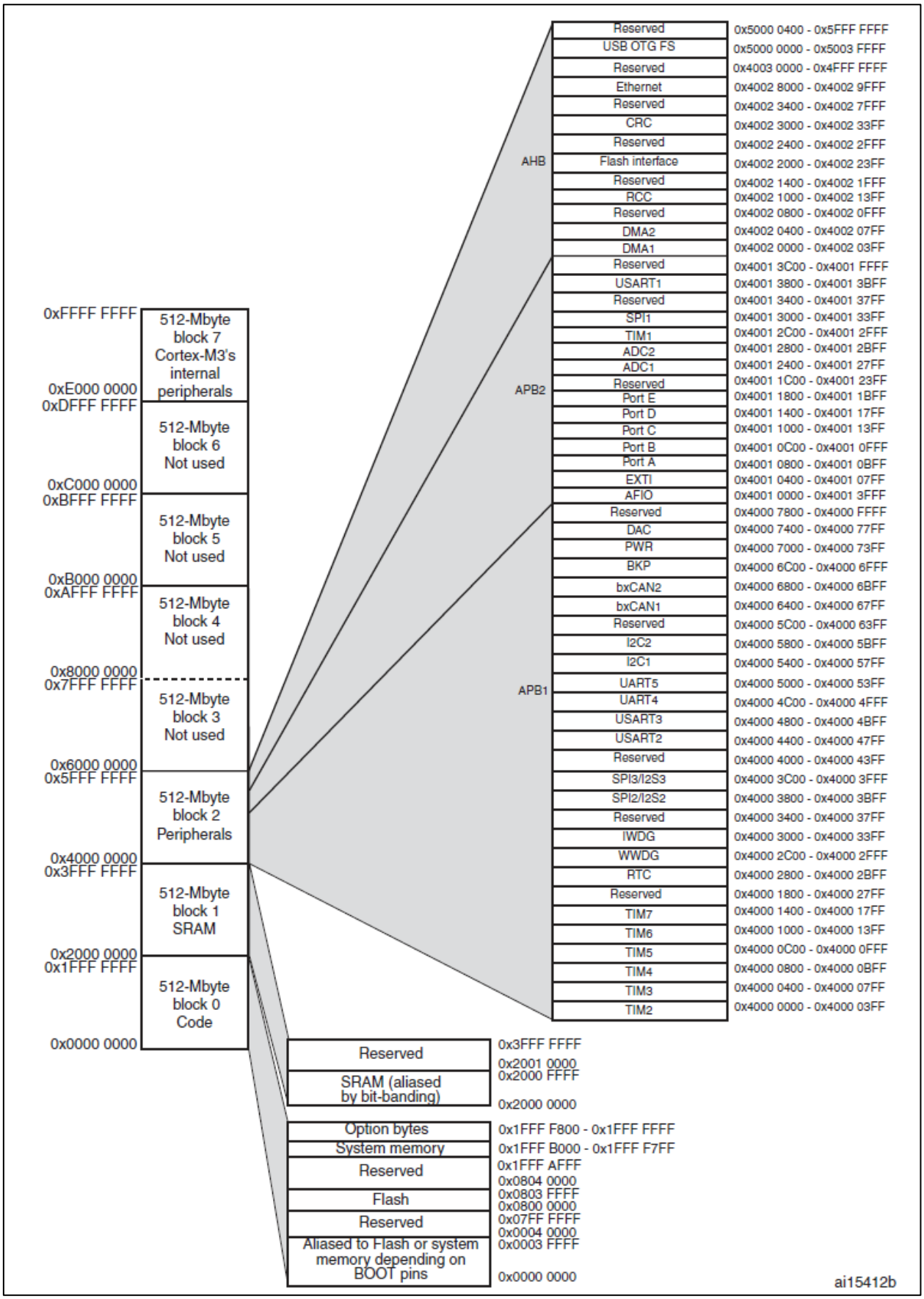

Рисунок 3 – Структура распределения адресного пространства МК

1.3.2 Выбор CAN-трансивера

CAN-трансивер представляет собой устройство обеспечивающее работу на физическом уровне шины CAN. Он предназначен для управления состоянием шины – установка доминантного состояния, поддержание рецессивного, чтение текущего состояния шины.

На рисунке 4 приведена временная диаграмма физических и логических уровней сигналов на шине CAN.

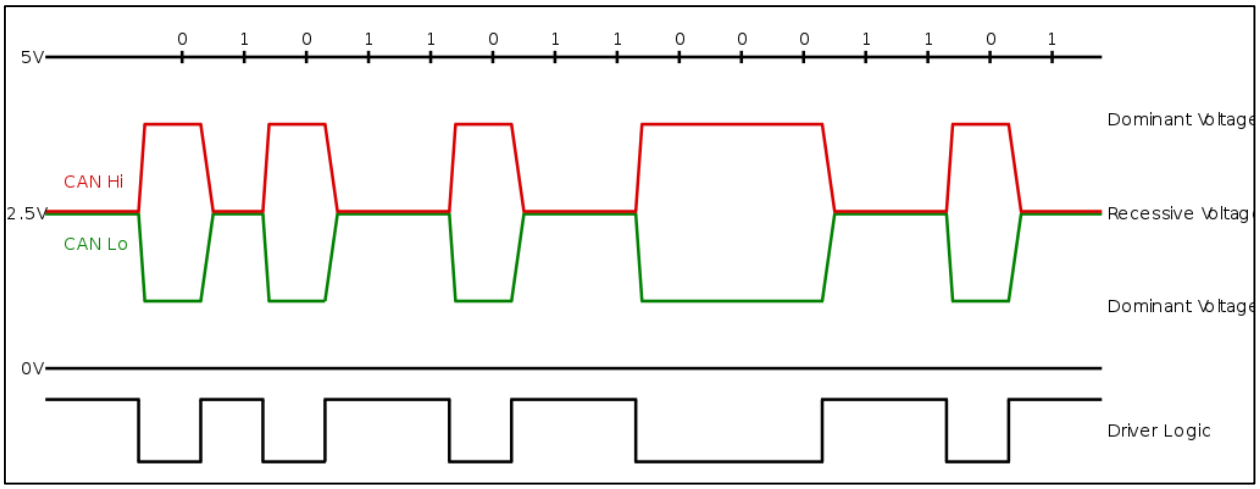

Рисунок 4 – Временная диаграмма шины CAN

Временная диаграмма стандартного CAN кадра с обозначением логического разделения содержимого представлена на рисунке 5.

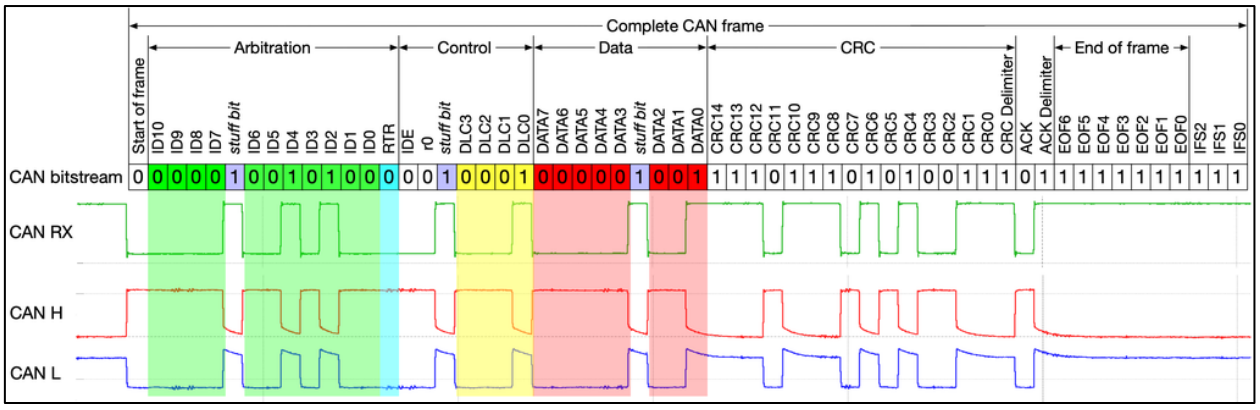

Рисунок 5 – Временная диаграмма стандартного кадра CAN

Основной характеристикой трансиверов является максимальная скорость работы с шиной. Также трансиверы могут включать в себе дополнительные функции: отключение передатчика, wake-up сигнал, защита шины от удержания доминантного состояния.

Большинство трансиверов представляют собой микросхему с восемью выводами и питанием +5 В.

В случае версии CAN 2.0B необходим трансивер, поддерживающий работу на скоростях до 1 Мбод/с.

В качестве CAN-трансивера была выбрана ИС TJA1050T/CM,118 производства NXP. Функциональная схема данной ИС представлена на рисунке 6 [5].

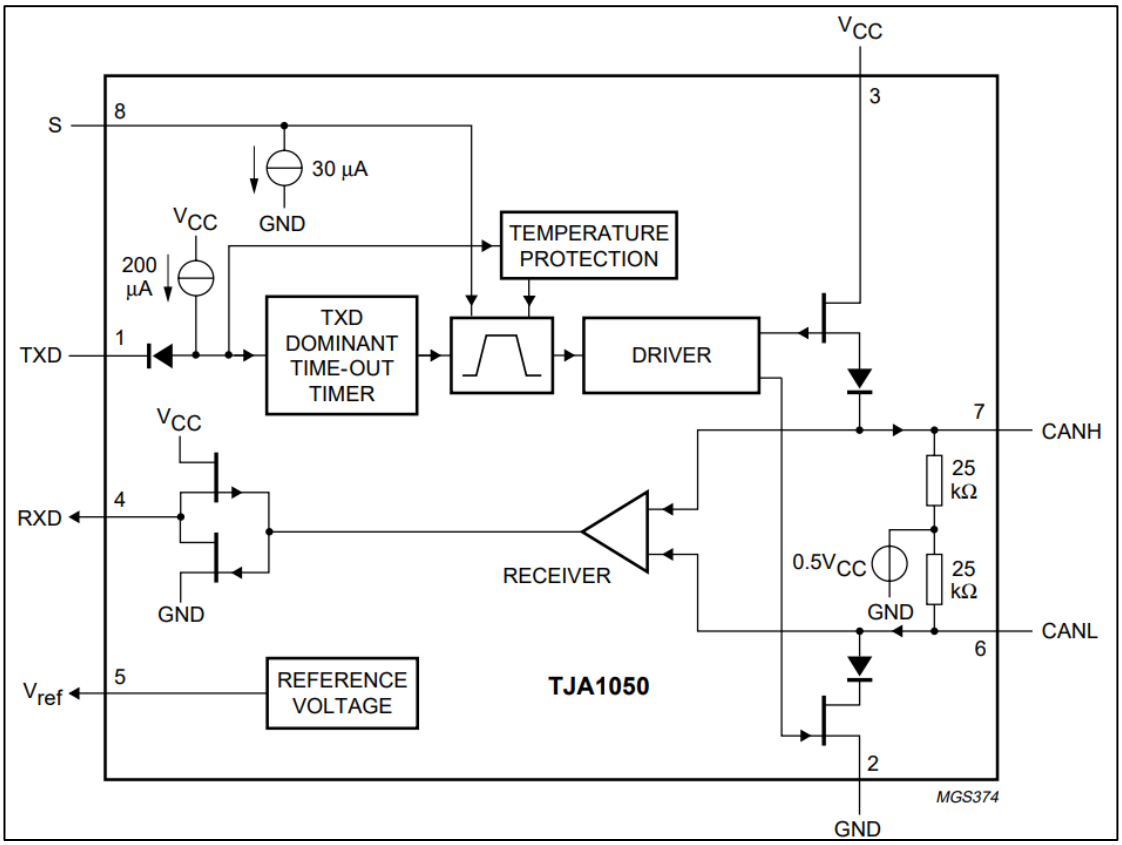

Рисунок 6 – Функциональная схема TJA1050T

1.3.3 Выбор компонентов цепей питания

Проектируемое устройство имеет возможность питания от +5 В USB, а также от БС автомобиля, напряжение которой на исправном автомобиле может варьироваться от 10,2 В до 14,8 В.

Для питания трансиверов CAN необходимо напряжение +5 В. Для питания МК требуется 3,3 В.

Общепринятым решением данной задачи будет использование понижающего (buck) [6] DC-DC преобразователя для преобразования высокого входного напряжения в +5 В. Выходной ток преобразователя будет

достаточен для питания трансиверов и последующего формирования напряжения +3,3 В для питания МК.

В качестве понижающего DC-DC преобразователя была выбрана ИС TPS563201 производства Texas Instruments.

+5 В также можно брать напрямую от USB интерфейса, подключив его через диод для защиты цепей ПЭВМ.

Для формирования напряжения +3,3 В целесообразно использовать LDO [7]. Несмотря на низкий КПД данных устройств, из-за низкого потребления МК мы можем использовать данное решение. В качестве LDO была выбрана ИС XC6206.

1.3.4 Выбор компонентов интерфейса USB

Выбранный МК поддерживает стандарт USB версии 2.0 FS и содержит контроллер физического уровня внутри себя. Внешний контроллер физического уровня USB 2.0 не требуется.

USB 2.0 представляет собой одну пару проводников, объединенных в дифференциальную пару. Устройства на шине USB делятся на 3 типа: «хост», «устройство» и «мост». Хостом на шине USB может являться единовременно лишь одно устройство. На рисунке 7 приведена диаграмма соединения устройств USB и их роли.

Передача данных всегда инициируется хостом, все подключенные устройства могут оправлять данные только в отведенные им хостом промежутки времени. На рисунке 8 приведена диаграмма процесса передачи данных по интерфейсу USB.

Компания-производитель МК STMicroelectronics рекомендует [8] защищать линии USB микроконтроллера от статических разрядов. Для этого производитель рекомендует использовать микросхему USBLC6-2.

1.3.5 Выбор разъемов устройства

Устройству необходимо по меньшей мере два разъема: для связи с автомобилем и с ПЭВМ, соответственно. Дополнительный разъем может быть предусмотрен для загрузки и отладки ПО микропроцессора.

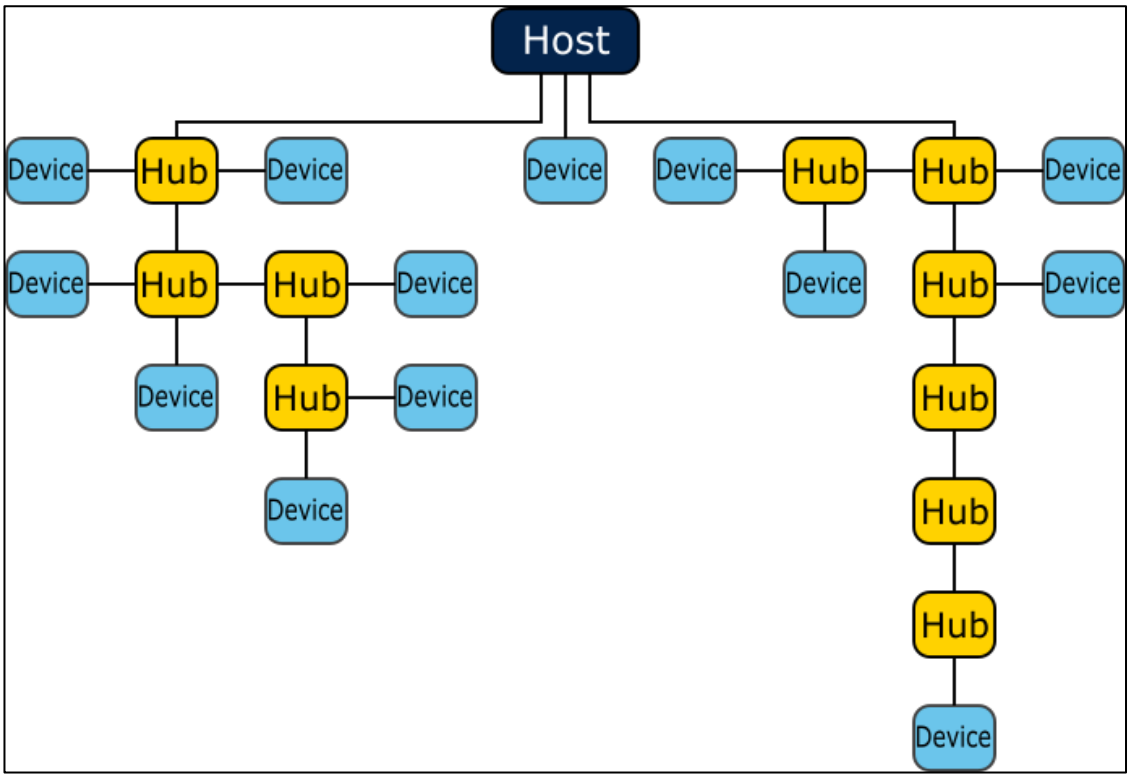

Рисунок 7 – Диаграмма соединения устройств USB в системе

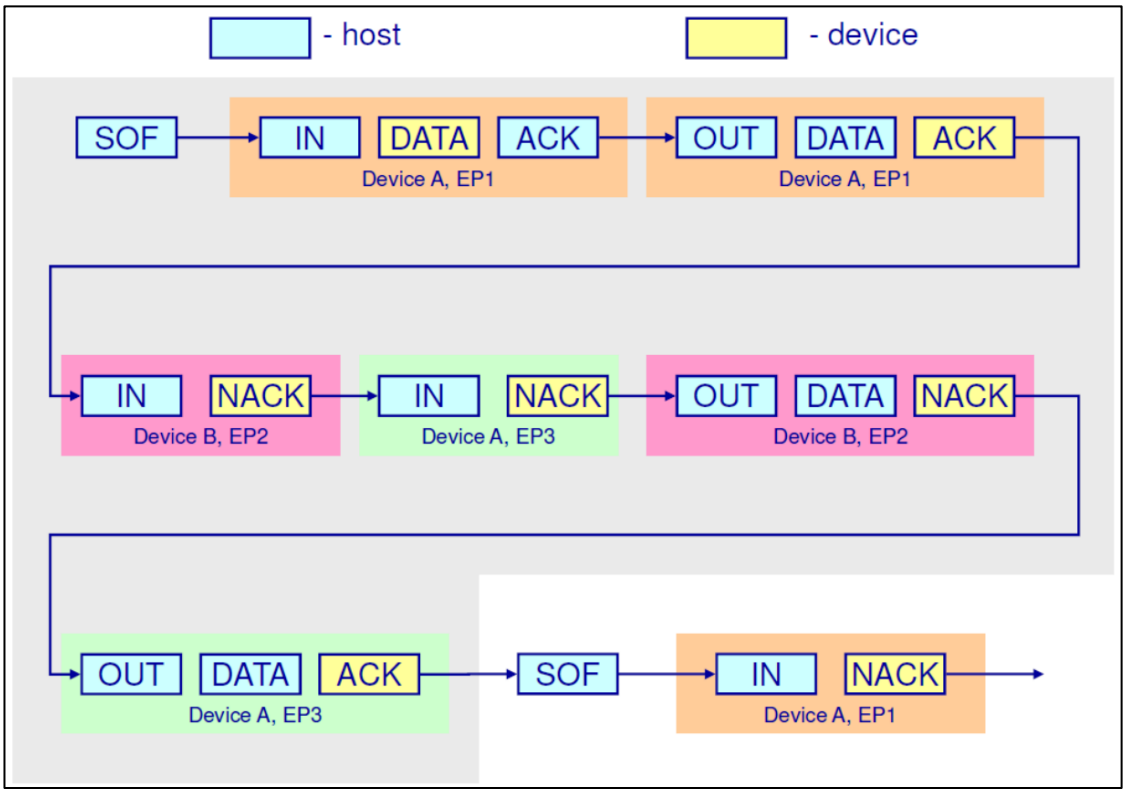

Рисунок 8 – Диаграмма передачи данных по USB

Соединение устройств USB может осуществляться с использованием стандартных разъемов, предусмотренных USB-IF [9]. Наиболее современным

и универсальным на момент проектирования устройства является разъем USB Type-C.

Данный тип разъема обладает следующими преимуществами:

– небольшие размеры относительно прочих разъемов,

– симметричен по горизонтали,

– имеет широкое распространение.

Подключение к автомобилю, согласно стандарту SAE J2534, должно осуществляться с использованием разъема SAE J1962 [10].

В качестве разъема для отладки и программирования для удобства использования решено использовать штыревую вилку PLS на 6 контактов.

1.4 Разработка электрической функциональной схемы

1.4.1 Цепи микроконтроллера

Для обеспечения работы микроконтроллера необходимо [11]:

– подключить соответствующие контакты к +3,3В и к земле,

– подключить цепи высокочастотного кварцевого резонатора,

– подключить цепи низкочастотного кварцевого резонатора,

– подключить подтягивающие резисторы ко входам сброса и выбора режима загрузки.

Для индикации текущего состояния устройства установим пару светодиодов, которыми сможет управлять МК.

Для корректной работы интерфейса USB подключим цепь +5 В от разъема USB через делитель напряжения к выделенному контакту МК [8].

Для удовлетворения требований стандарта SAE J2534 необходимо обеспечить возможность замера напряжения БС автомобиля. Подключим цепь +12 В через делитель напряжения к одному из входов АЦП микроконтроллера.

1.4.2 Цепи CAN

Для работы CAN-трансиверов необходимо подключить к соответствующим контактам +5 В и землю.

Для передачи данных по шине CAN необходимо подключить выводы RX и TX к соответствующим контактам МК. Каждый трансивер подключается к отдельному контроллеру CAN, встроенному в МК.

Также подключим вывод перехода в «тихий» режим к МК, чтобы иметь возможность принудительно работать в режиме «только чтение».

Выводы CANH и CANL необходимо подключить к контактам 3, 11 и 6, 14 разъема J1962 для каждого трансивера соответственно.

1.4.3 Цепи питания

Для формирования напряжения +5 В используется DC-DC преобразователь, построенный на базе ИС TPS563201. Помимо самой ИС для формирования +5 В необходима обвязка для неё. В неё входят следующие части [12]:

- катушка индуктивности,
- делитель напряжения,
- входные и выходные емкости.

Вход DC-DC преобразователя подключен к +12 В от БС автомобиля посредством контакта 16 разъема J1962. Земля устройства объединена с массой автомобиля через контакты 4 и 5 разъема J1962.

Также к цепи +5 В через диод Шоттки подключена шина +5 В USB, чтобы обеспечить возможность питания от USB и уберечь ПЭВМ от возможных сбоев в работе цепей питания устройства.

Для формирования напряжения +3,3 В используется LDO XC6206, подключаемое к цепи +5 В. Это позволит питать МК как при подключении к автомобилю, так и при питании от USB-разъема, где происходит небольшое падение напряжения на диоде Шоттки.

1.4.4 Цепи интерфейса USB

Для корректной работы разъема USB Type-C необходимо подключить контакты CC1 и CC2 согласно стандарту [9].

Так как выбранный разъем симметричен и имеет поддержку USB 2.0, то и контактов D+ и D- у него две пары. Согласно стандарту, единовременно

может быть подключена только одна пара, вторая остается незадействованной, так как соединение отсутствует на уровне кабеля.

Дифференциальную пару USB 2.0 подключим к МК через ИС защиты от статических разрядов.

1.4.5 Цепи разъема отладки

Для обеспечения возможности загрузки прошивки и отладки устройства необходимо подключить линии интерфейса отладки МК к разъему.

Микроконтроллер STM32F105RB предоставляет два интерфейса отладки: SWD и JTAG.

Интерфейс SWD работает по двум линиям: SWCLK и SWDIO. Дополнительно может быть подключена линия SWO.

Интерфейс JTAG работает по четырем линиям: JTCK, JTMS, JTDI, JTDO.

Линии SWD и JTAG скоммутированы внутри МК и могут быть переназначены в процессе работы, поэтому для подключения этих интерфейсов требуется лишь четыре контакта, помимо питания и земли.

1.4.6 Итоговая электрическая функциональная схема

Полученная в результате работы электрическая функциональная схема приведена в приложении А.

1.5 Разработка электрической принципиальной схемы

1.5.1 Расчет цепей тактирования микропроцессора

Расчет цепей кварцевых резонаторов осуществлялся в соответствии с рекомендациями производителя МК [13].

В качестве высокоскоростного кварцевого резонатора выбран резонатор TAXM8M4RDBCCT2T с частотой 8 МГц. Данная частота тактирования наиболее удобна для дальнейшей настройки цепей тактирования внутри МК.

Емкость нагрузочных конденсаторов была рассчитана исходя из нагрузочной способности высокочастотного резонатора [14] и емкостей выводов МК [4]. Емкость нагрузочных конденсаторов составляет 10 пФ.

В качестве низкоскоростного кварцевого резонатора был выбран резонатор NX3215SA с частотой 32,768 кГц.

Емкость нагрузочных конденсаторов была рассчитана исходя из нагрузочной способности низкочастотного резонатора [15] и емкостей выводов МК [4]. Емкость нагрузочных конденсаторов составляет 12 пФ.

Также в цепь между выходным контактом МК и кварцевым резонатором был поставлен резистор нулевого сопротивления, для возможности дальнейшей подстройки.

1.5.2 Расчет цепей развязочных конденсаторов

Развязка цепей питания МК выполнялась согласно рекомендациям производителя [11].

Для развязки основного питания процессора используется по одному керамическому конденсатору номиналом 100 нФ на каждый контакт и один общий более емкий на 1 мкФ.

Для развязки питания аналоговой части МК используются керамические конденсаторы емкостями 100 нФ и 1 мкФ.

Для развязки питания CAN-трансиверов используется по одному керамическому конденсатору номиналом 100 нФ на каждый.

На входе и выходе LDO, формирующего +3,3 В, установлены керамические развязывающие конденсаторы номиналом 1 мкФ.

1.5.3 Расчет делителей напряжения

В устройстве предусмотрено два делителя напряжения: для обнаружения USB подключения и замера напряжения БС автомобиля. АЦП микроконтроллера способен измерять напряжение в диапазоне от 0 В до напряжения питания. Следовательно, нам необходимо задать такие параметры для делителей напряжения, чтобы напряжение никогда не превышало напряжение питания.

Номиналы резисторов делителя напряжения для обнаружения USB подключения взяты в соответствии с рекомендациями производителя МК [8], по 33 кОм и 82 кОм для верхнего и нижнего плеча соответственно.

Номиналы резисторов делителя напряжения для замера напряжения БС автомобиля рассчитаны так, чтобы напряжение, поступающее на контакт МК не было выше 3,3 В. Максимальное напряжение БС автомобиля принято за 17 В.

Коэффициент деления делителя напряжения рассчитывается по следующей формуле:

$$К_д = \frac{R_н}{R_в + R_н},$$

где $R_в$ *– номинал верхнего плеча,* $R_н$ *– номинал нижнего плеча.*

Выходное напряжение рассчитывается по следующей формуле:

$$U_{вых} = \frac{U_{вх}}{К_д}$$

Отсюда получаем формулу расчета необходимого коэффициента деления:

$$К_д = \frac{U_{вх}}{U_{вых}}$$

Подставляя в данную формулу исходные данные, получаем $К_д = 5,(15)$. Далее выбираем из стандартного ряда пару номиналов с наиболее близким соотношением напряжений. Допустимо незначительное отклонение коэффициента деления. Суммарное сопротивление делителя определяет ток, протекающий через делитель, от чего зависит влияние тока потребления МК на точность измерений.

Резистор верхнего плеча выбран номиналом 10 кОм, резистор нижнего плеча выбран номиналом 2,49 кОм с итоговым коэффициентом деления равным 5,02.

1.5.4 Расчет токоограничивающих и подтягивающих резисторов

Согласно информации [4], предоставленной производителем МК, максимальный суммарный ток, который микроконтроллер может отдать и поглотить – 150 мА в каждую сторону. Максимальный ток на одном контакте, который микроконтроллер может отдать и поглотить – 25 мА.

Чтобы не создавать излишнее потребление энергии микроконтроллером, ограничим ток потребления светодиодных индикаторов на уровне 5 мА.

В качестве индикаторов работы были выбраны светодиоды с падением напряжения при токе в 5 мА равным 2,9 В. Расчет номинала сопротивления производим по закону Ома, где в качестве напряжения выступает разность напряжения питания и падения напряжения на светодиоде. Получаем резисторы с номиналом 80 Ом.

Подтягивающие резисторы для контактов определяющих режим загрузки МК выбраны номиналом 10 кОм.

1.5.5 Расчет цепей DC-DC преобразователя

Расчет номиналов элементов для цепей DC-DC преобразователя производился согласно информации, предоставленной производителем [12]. На рисунке 9 представлена типовая схема подключения ИС TPS563201.

Согласно таблице рекомендуемых номиналов компонентов, предоставленной производителем, при выходном напряжении 5 В следует использовать компоненты следующих номиналов:

– C1, C2 – 10 мкФ; C7 – 100 нФ; C8, C9 – 22 мкФ,

– L1 – 3,3 мкГн,

– R1 – 54,9 Ом; R2, R3 – 10 кОм.

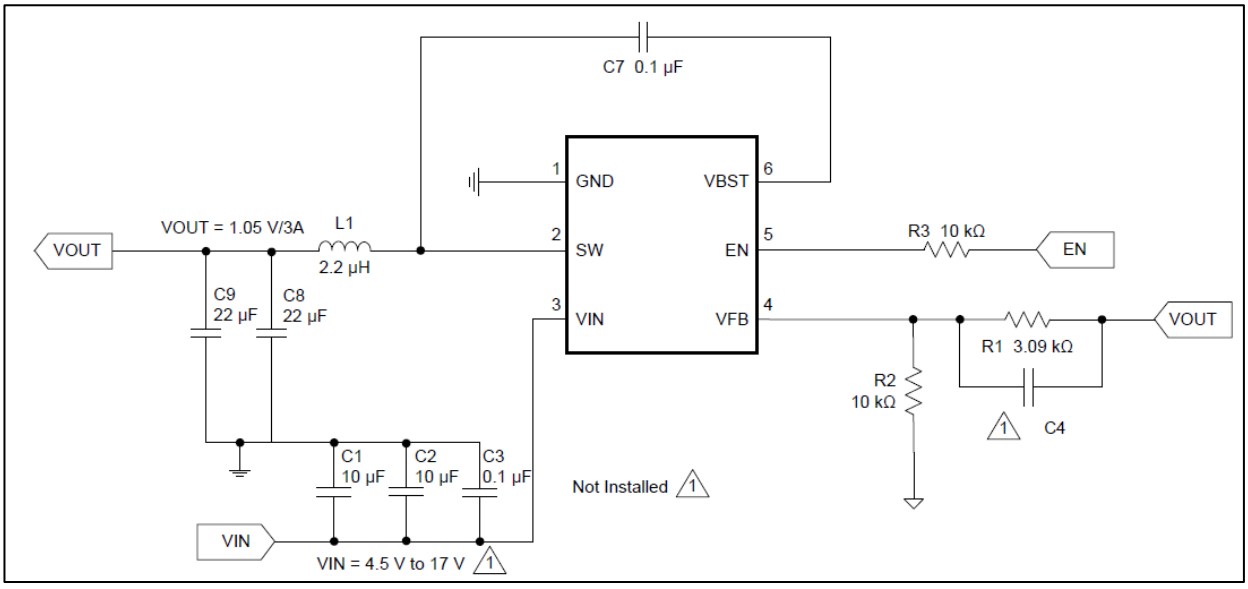

Рисунок 9 – Типовая схема подключения TPS563201

Одними из параметров данной схемы, требующими отдельного расчета, являются I_{PEAK} и $I_{LO(RMS)}$. Подбор катушки индуктивности должен осуществляться с их учетом. Для расчета данных параметров потребуются следующие формулы:

$$I_{P-P} = \frac{V_{OUT}}{V_{IN(MAX)}} * \frac{V_{IN(MAX)} - V_{OUT}}{L_O * f_{SW}},$$

где $V_{IN(MAX)}$ – максимальное входное напряжение, V_{OUT} – желаемое выходное напряжение, L_O – индуктивность выходной катушки, f_{SW} – частота работы ключа.

$$I_{PEAK} = I_O + \frac{I_{P-P}}{2},$$

где I_O – максимальный выходной ток.

$$I_{LO(RMS)} = \sqrt{{I_O}^2 + \frac{{I_{P-P}}^2}{12}}$$

V_{OUT} в нашем устройстве равен 5 В, $V_{IN(MAX)}$ равен 16 В, L_O равен 3,3 мкГн, f_{SW} равен 580 кГц, I_O примем за 750 мА.

По итогам расчетов имеем $I_{PEAK} = 1{,}65$ А, $I_{LO(RMS)} = 0{,}912$ А.

Значит катушка индуктивности должна быть номиналом 3,3 мкГн, иметь $I_{sat} \geq 1{,}65$ А и $I_{RMS} \geq 0{,}912$ А.

1.5.6 Расчет цепей разъема USB Type-C

Изначально разъем USB Type-C имеет 24 контакта, но так как для нашего устройства необходим только стандарт USB 2.0 и питание, то мы можем воспользоваться упрощенным вариантом этого разъема с меньшим количеством выводов.

Подключаем дифференциальную пару к МК через защиту от электростатических разрядов. А также подключаем выводы разъема CC1 и CC2 к земле через резисторы 5,1 кОм согласно стандарту [9].

1.5.7 Итоговая электрическая принципиальная схема

Полученная в результате работы электрическая принципиальная схема приведена в приложении Б.

1.6 Проектирование печатной платы

1.6.1 Общая сводка

Печатная плата устройства проектируется под имеющийся корпус с разъемом J1962. Фотография корпуса приведена на рисунке 10.

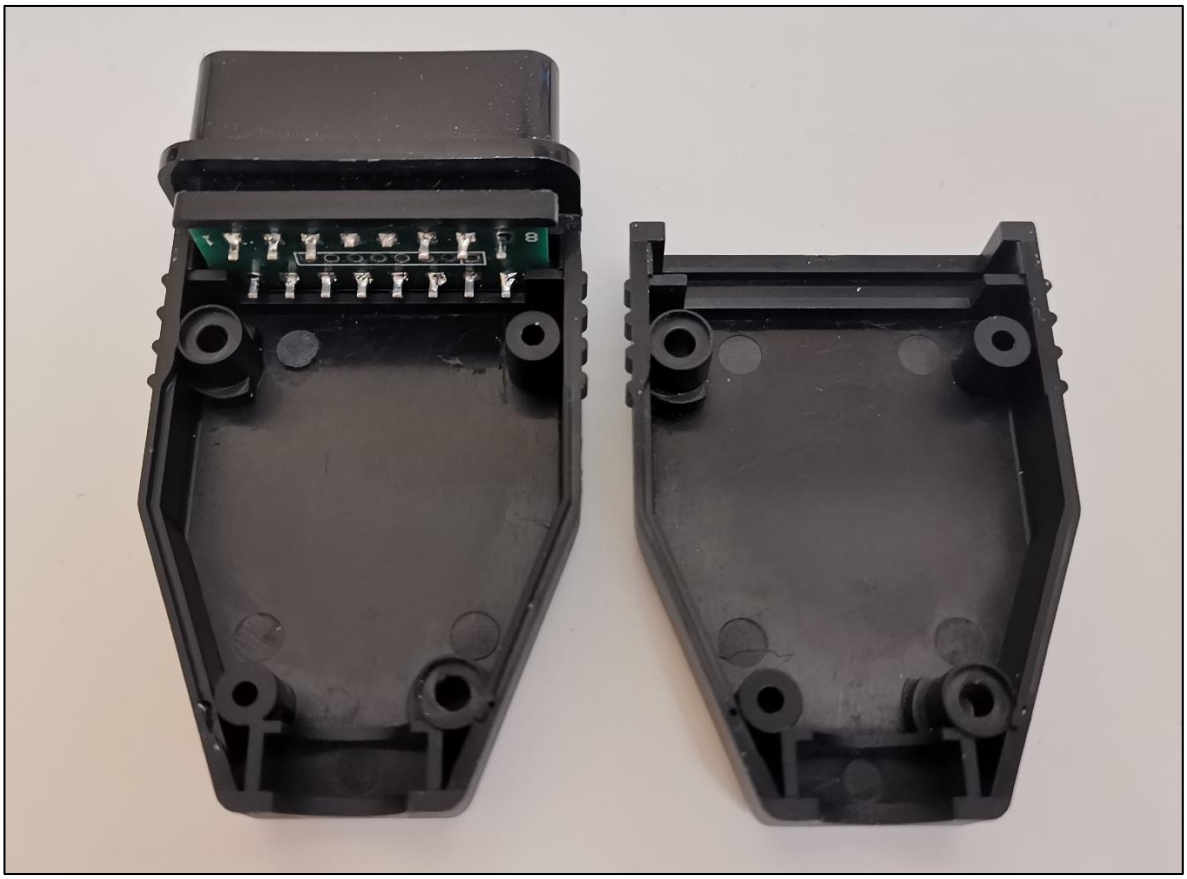

Рисунок 10 – Корпус устройства

Из-за малых размеров печатной платы, было принято решение проектировать четырехслойную печатную плату. Это позволит уменьшить воздействие электромагнитных помех на цепи устройства.

При проектировании печатной платы в качестве обучающих материалов были использованы различные информационные интернет-ресурсы [16–19].

1.6.2 Итог проектирования печатной платы

Результаты проектирования печатной платы в виде изображений трехмерной модели устройства: изометрия, вид сверху и вид снизу приведены на рисунках 11-12. Изображения верхних и нижних слоев печатной платы приведены на рисунке 13. Изображения внутренних слоёв не приводятся, так как представляют собой тривиальные слои с цепями питания устройства.

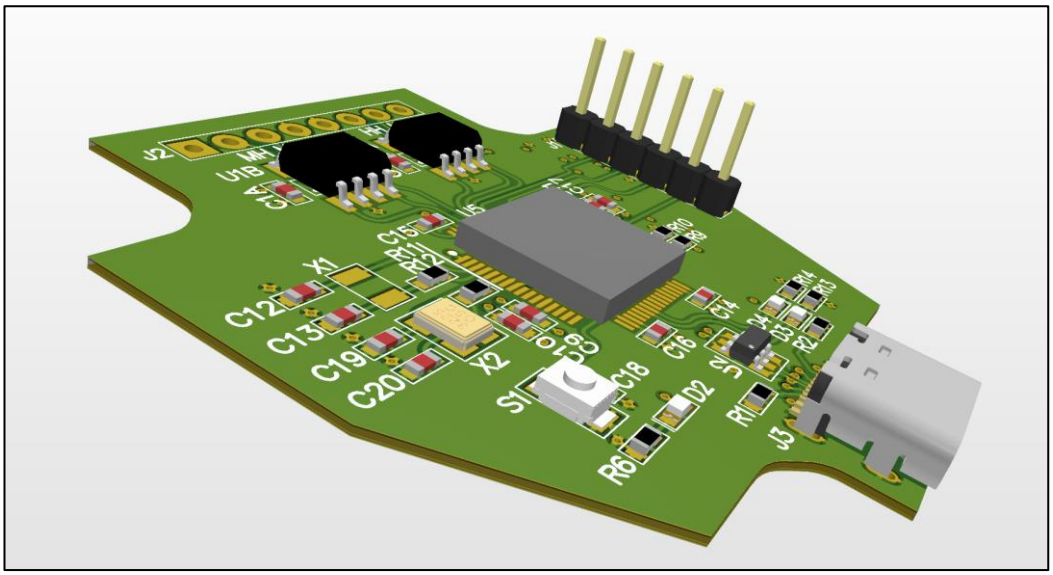

Рисунок 11 – Изометрический вид

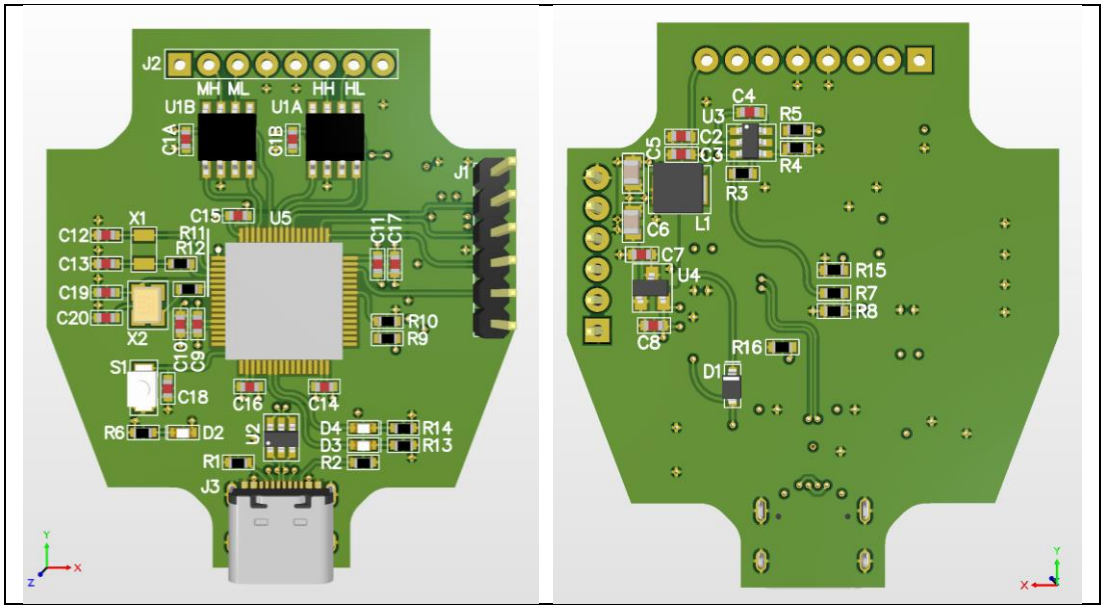

Рисунок 12 – Вид сверху и вид снизу

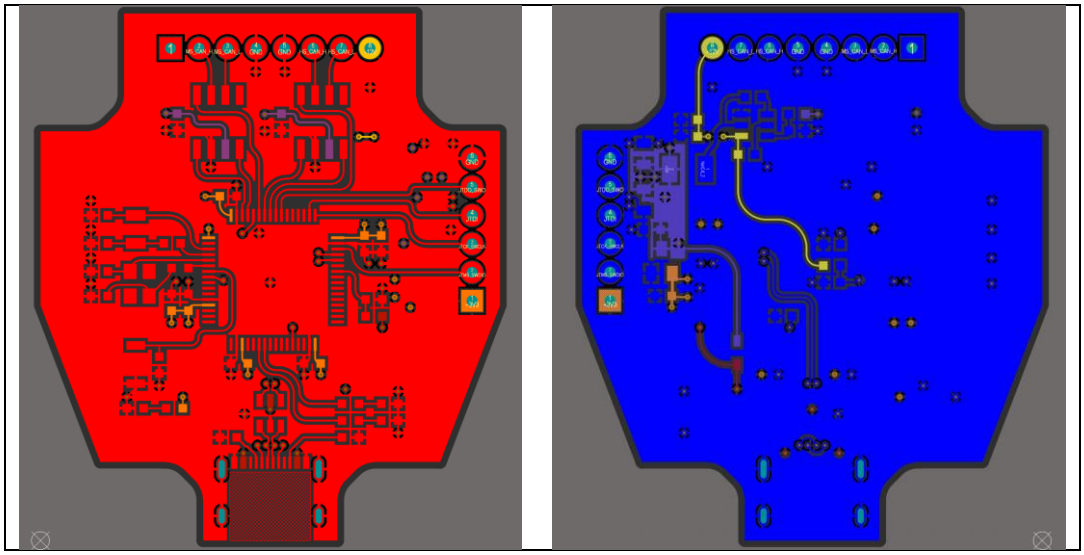

Рисунок 13 – Верхний и нижний слои печатной платы

1.7 Расчет потребляемой мощности

Для расчета потребляемой мощности необходимо определить потребление тока каждого узла устройства и его напряжение питания.

Потребляемую мощность ИС возьмем из соответствующих документов. Для подсчета потребления микроконтроллера учтем, что он работает на частоте 72 МГц.

Для определения потребляемой мощности аналоговыми элементами воспользуемся следующей формулой:

$$P_{\text{пот}} = U_{\text{пад}} * I_{\text{пот}}$$

В таблице 2 приведены потребляемые составными частями устройства мощности.

Таблица 2 – Мощности, потребляемые составными частями устройства

Узел	Кол-во	Напряжение, В	Ток, мА	Мощность, мВт
STM32F105RB	1	3,3	47,3 [4]	156,09
TJA1050T	2	5	5 [5]	50
Светодиодный индикатор	3	3,3	5	49,5
Делитель напряжения USB	1	5	0,05	0,25
Делитель напряжения БС	1	12,5	1	12,5

Суммарное потребление $P_{\text{сум}} = 268{,}34$ мВт.

1.8 Разработка ПО устройства

1.8.1 Логическая структура программы

Согласно стандарту SAE J2534 производитель устройства обязан предоставить интерфейс взаимодействия ПЭВМ с диагностическими шинами автомобиля напрямую, без учета специфики какой-либо диагностической шины. Стандарт строго не регламентирует физический интерфейс

взаимодействия с устройством, но описывает логический интерфейс, его функции и передаваемые параметры.

Так как стандарт предусматривает возможность взаимодействия устройства посредством различных интерфейсов передачи данных, целесообразно разработать единый программный интерфейс на уровне устройства для всех подобных интерфейсов.

Обработка запросов стандарта SAE J2534 производится единым образом для каждого вида диагностических шин автомобиля в модуле обработки команд.

Определение возможности применения конкретной функции к конкретному каналу передачи данных определяется модулем интерфейса данного канала.

1.8.2 Модуль интерфейса

Модуль интерфейса преобразует запросы, передаваемые через конкретный интерфейс передачи данных, в запросы единого формата для модуля обработки данных.

Модуль интерфейса должен быть запрограммирован отдельно для каждого конкретного интерфейса передачи данных, так как каждый из них предусматривает собственный способ передачи команд устройству.

В ходе данной работы был реализован модуль интерфейса, реализующий обработку команд, поступающих через интерфейс USB 2.0. Для передачи данных по USB был использован класс VCP, как наиболее легкий для взаимодействия с ним пользователя, а не программы.

Он позволяет отправлять команды и принимать ответы устройства используя программу-терминал, позволяющую взаимодействовать с COM-портами ПЭВМ.

Схемы алгоритмов модуля интерфейса приведены на рисунке 14.

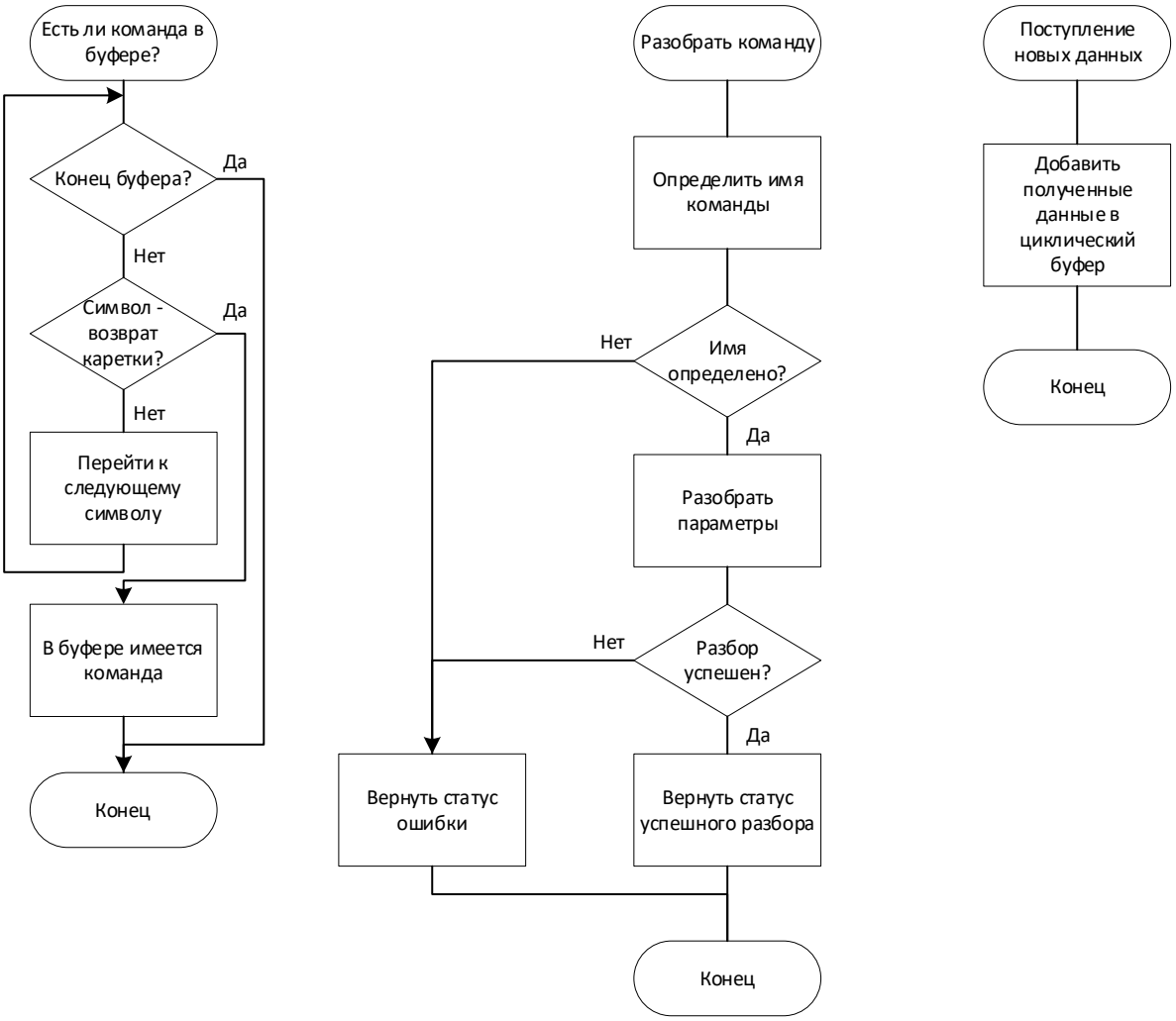

Рисунок 14 – Схема алгоритма работы интерфейса VCP

1.8.3 Модуль обработки команд

Модуль обработки команд хранит в себе структуры модуля интерфейса взаимодействия и модулей интерфейсов диагностических шин. Он осуществляет передачу команд и возвращаемых значений между данными интерфейсами. Также он осуществляет первичную инициализацию данных модулей и последующий контроль за их работой с обработкой ошибок в случае их возникновения.

Данный модуль абстрагирован от конкретной реализации тех или иных модулей посредством общих интерфейсов взаимодействия с ними. Программист может реализовывать дополнительные интерфейсы без необходимости изменять код модуля обработки команд.

Схема алгоритма модуля обработки команд приведена на рисунке 15.

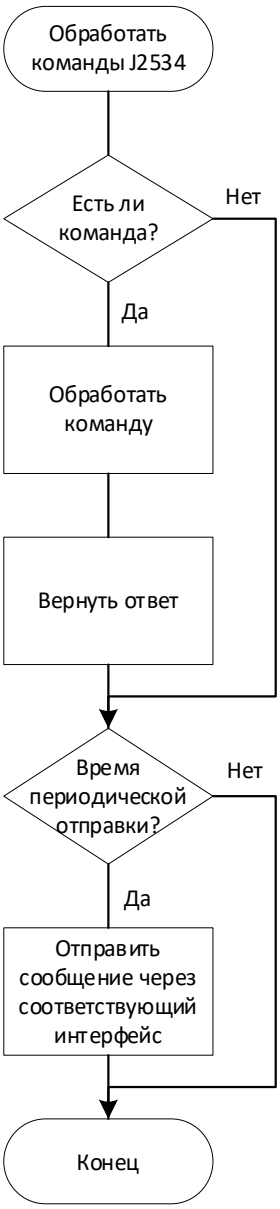

Рисунок 15 – Схема алгоритма модуля обработки команд

1.8.4 Модуль интерфейса CAN

Модуль интерфейса CAN обрабатывает запросы настройки, передачи и приема данных от модуля обработки команд и обслуживает прерывания, вызываемые модулем CAN-контроллера. Таким образом он обеспечивает взаимодействие ПЭВМ с шиной CAN в автомобиле.

Схемы алгоритмов модуля интерфейса CAN приведены на рисунке 16.

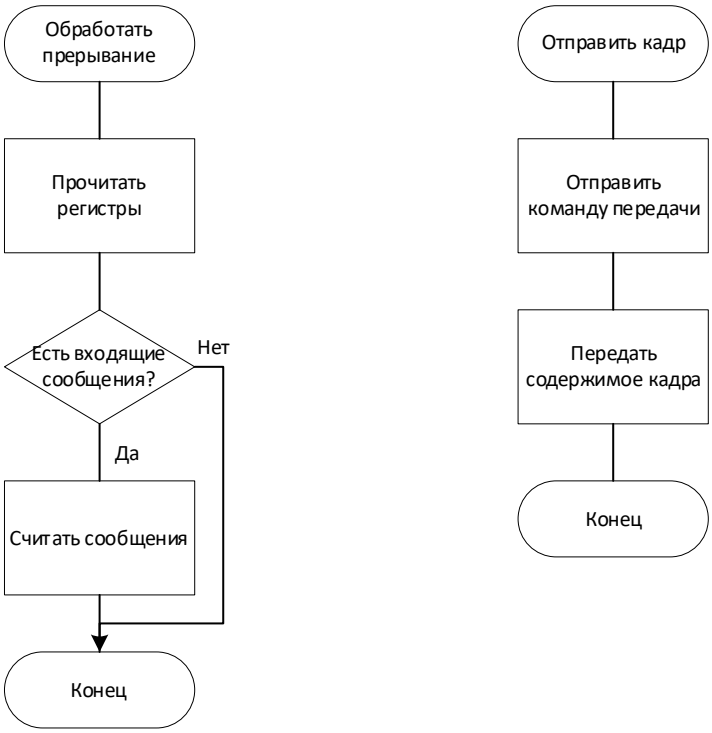

Рисунок 16 – Схема алгоритма модуля интерфейса CAN

1.9 Интерфейс взаимодействия с устройством

Интерфейс взаимодействия с устройством реализован посредством отправки символьных строк через терминал COM-порта. В случае успешного выполнения команды устройство вернет ответ или сообщение об успешном выполнении, либо вернет код ошибки, которые определены в стандарте SAE J2534.

Список доступных команд и их параметры определены в стандарте, а формат их передачи на устройство приведен ниже:

<имя команды>

<имя команды> <параметр1>,<параметр2>, … , <параметр N>

Имя команды отделяется от передаваемых параметров пробелом, если таковые параметры имеются. Параметры разделяются между собой символом запятой. Конец команды обозначается символом возврата каретки с кодом 13.

Список имен команд приведен далее, а примеры использования команд описаны в приложении Г: `connect`, `disconnect`, `readmsgs`, `writemsgs`, `startperiodicmsgs`, `stopperiodicmsgs`, `startmsgfilter`, `stopmsgfilter`, `setprogvoltage`, `readversion`, `getlasterror`, `ioctl`.

2 Технологическая часть

2.1 Используемые инструменты

При разработке устройства были использованы следующие программные средства:

– STM32CubeIDE – среда программной разработки для МК STM32,

– Altium Designer – САПР для проектирования радиоэлектронных средств,

– sPlan – программа для черчения схем и чертежей.

Из аппаратных средств использовались следующие:

– отладочная плата на базе МК STM32F401CC,

– программатор ST-link v2,

– модули шины CAN с контроллером MCP2515.

2.2 Методы тестирования

Тестирование и отладка программного кода проводились с использованием макетной платы и программатора ST-link. На рисунке 17 изображена макетная плата, на которой производилась отладка ПО.

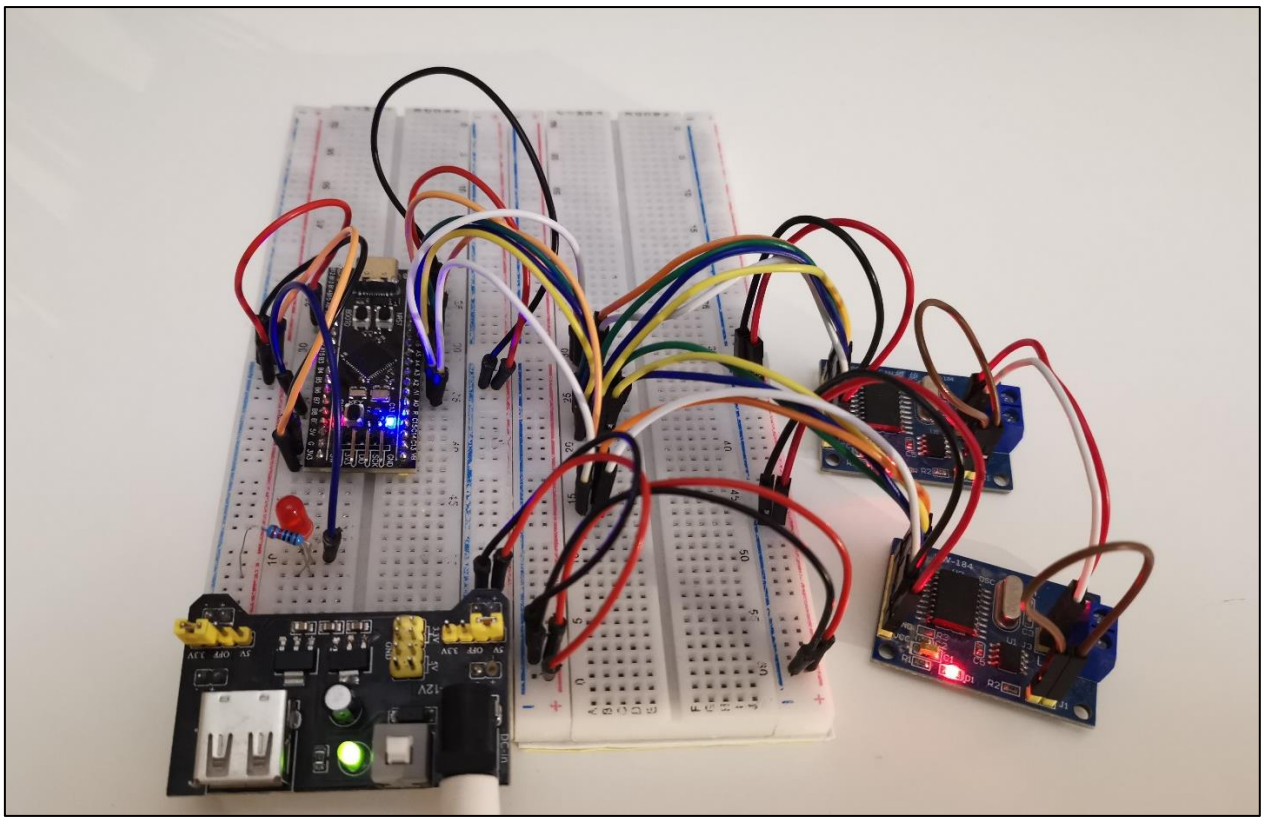

Рисунок 17 – Отладочная макетная плата

Ядра ARM Cortex имеют встроенный интерфейс отладки с возможностью установки аппаратных и программных точек останова. Используя среду разработки STM32CubeIDE и программатор ST-link имеется возможность пошаговой отладки кода, просмотра содержимого регистров и значений переменных прямо на микроконтроллере.

Методом тестирования была выбрана ручная проверка корректности исполнения функций. Микроконтроллеру отправляются тестовые данные, ставится две точки останова: до входа в функцию и после выхода из неё. Две точки необходимы чтобы проверить корректность входных и выходных данных соответственно. Если была обнаружена ошибка в работе функции, тест перезапускается и производится полная пошаговая отладка функции с теми же входными данными. Пример проводимых тестов ПО устройства представлен на рисунке 18.

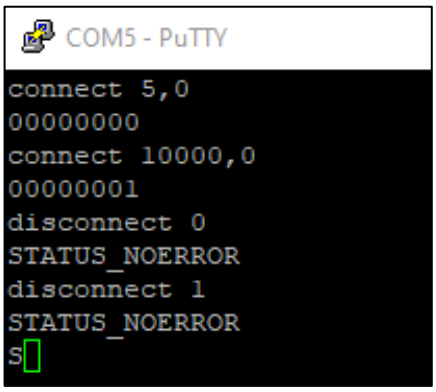

Рисунок 18 – Тестирование программного обеспечения

2.3 Описание способа программирования

При разработке ПО устройства применялся метод внутрисхемного программирования, при котором для загрузки программного обеспечения МК не требуется его демонтаж с печатной платы.

Интерфейсом программирования выступал интерфейс SWD [20], который представляет собой в минимальной конфигурации две линии: линию тактирования и двунаправленную линию передачи данных. Также может быть применена отдельная линия для вывода отладочной информации во время работы МК.

Данный интерфейс является частью ядра ARM Cortex, что позволяет получить доступ к большинству ресурсов микроконтроллера. Функциональная диаграмма интерфейса SWD совмещенного с JTAG приведена на рисунке 19.

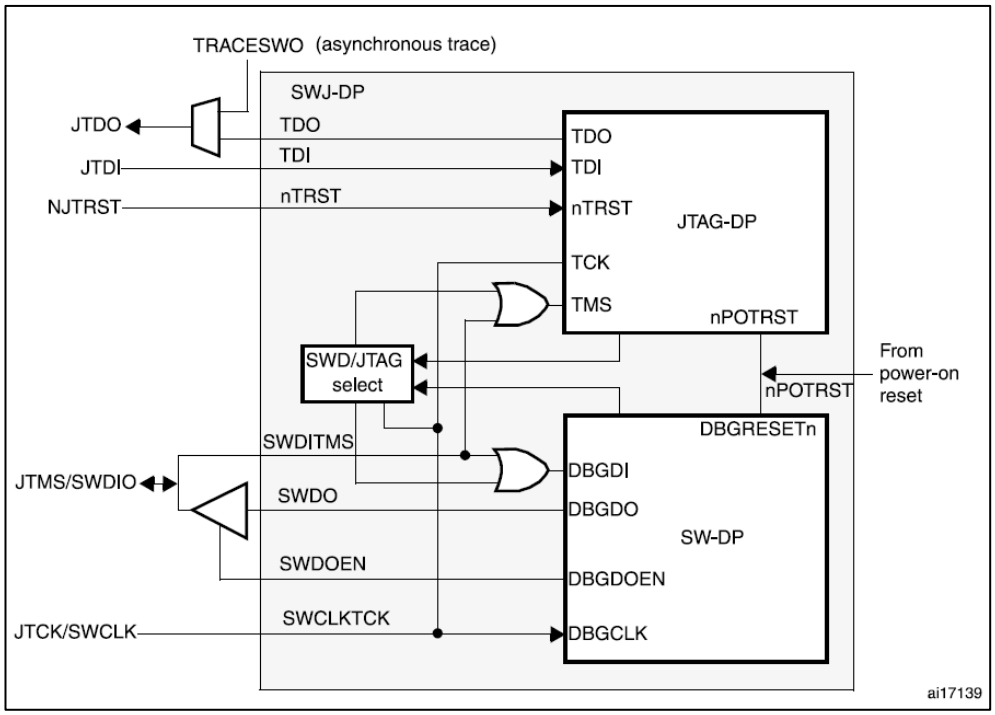

Рисунок 19 – Функциональная диаграмма интерфейса отладки

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано устройство-адаптер. Данное устройство способно взаимодействовать с ПЭВМ согласно стандарту SAE J2534; имеет интерфейс CAN, подключаемый к автомобилю через разъем J1962 и способно обеспечить взаимодействие ПЭВМ с диагностической шиной CAN автомобиля.

В результате проектирования разработаны электрические функциональная и принципиальная схемы, спроектирована печатная плата, а также написано и отлажено программное обеспечение устройства.

В результате проведенной работы спроектировано устройство, удовлетворяющее требованиям технического задания.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. SAE J2534:2002. Recommended Practice for Pass-Thru Vehicle Programming — URL: https://www.sae.org/standards/content/j2534_200202/ (дата обращения: 04.09.2022).

2. ISO 11898-2:2016 Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit. — URL: https://www.iso.org/standard/67244.html (дата обращения 17.09.2022).

3. ISO 15765-4:2021. Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) — Part 4: Requirements for emissions-related systems. — URL: https://www.iso.org/standard/78384.html (дата обращения 17.09.2022).

4. STMicroelectronics. DS6014. STM32F105XX, STM32F107XX Datasheet. — URL: https://www.st.com/resource/en/datasheet/stm32f105rb.pdf (дата обращения 03.10.2022).

5. NXP. TJA1050T High speed CAN transceiver. Datasheet — URL: https://static.chipdip.ru/lib/205/DOC000205042.pdf (дата обращения 04.10.2022).

6. Texas Instruments. Linear and Switching Voltage Regulator Fundamentals — URL: https://www.ti.com/lit/an/snva558/snva558.pdf (дата обращения 25.09.2022).

7. Texas Instruments. LDO Basics. — URL: https://www.ti.com/lit/eb/slyy151a/slyy151a.pdf (дата обращения 25.09.2022).

8. STMicroelectronics. AN4879. USB hardware and PCB guidelines using STM32 MCUs. — URL: https://www.st.com/resource/en/application_note/dm00296349-usb-hardware-and-pcb-guidelines-using-stm32-mcus-stmicroelectronics.pdf (дата обращения 20.09.2022).

9. USB Implementers Forum Official Website. — URL: https://www.usb.org/ (дата обращения 15.10.2022).

10. SAE J1962:2002. Diagnostic Connector. — URL: https://www.sae.org/standards/content/j1962_201607/ (дата обращения 04.10.2022).

11. STMicroelectronics. AN2586. Getting started with STM32F10xxx hardware development. — URL: https://www.st.com/content/ccc/resource/technical/document/application_note/6c/a3/24/49/a5/d4/4a/db/CD00164185.pdf/files/CD00164185.pdf/jcr:content/translations/en.CD00164185.pdf (дата обращения 15.10.2022).

12. Texas Instruments. TPS56320x Datasheet. — URL: https://www.ti.com/lit/gpn/tps563208 (дата обращения 16.11.2022).

13. STMicroelectronics. AN2867. Oscillator design guide for STM8AF/AL/S, STM32 MCUs and MPUs. — URL: https://www.st.com/resource/en/application_note/cd00221665-oscillator-design-guide-for-stm8afals-stm32-mcus-and-mpus-stmicroelectronics.pdf (дата обращения 23.10.2022).

14. TAXM8M4RDBCCT2T Specifications. — URL: https://datasheet.lcsc.com/lcsc/1912111437_TAE-Zhejiang-Abel-Elec-TAXM8M4RDBCCT2T_C400090.pdf (дата обращения 25.10.2022).

15. NX3215SA Specifications — URL: https://datasheet.lcsc.com/lcsc/1810171130_NDK-NX3215SA-32-768K-STD-MUA-14_C156244.pdf (дата обращения 25.10.2022).

16. Altium Education. — URL: https://education.altium.com/ (дата обращения 24.09.2022).

17. Altium Academy Youtube channel. — URL: https://www.youtube.com/@AltiumAcademy (дата обращения 14.11.2022).

18. EEVblog Youtube channel. — URL: https://www.youtube.com/@EEVblog (дата обращения 13.11.2022).

19. Vladimir Medintsev Youtube channel — URL: https://www.youtube.com/@VladimirMedintsev (дата обращения 15.10.2022).

20. STMicroelectronics. RM0008. Reference manual. STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs — URL: https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf (дата обращения 18.09.2022).

ПРИЛОЖЕНИЕ А

Схема электрическая функциональная

ПРИЛОЖЕНИЕ Б

Схема электрическая принципиальная

ПРИЛОЖЕНИЕ В

Перечень элементов

ПРИЛОЖЕНИЕ Г

Описание интерфейса взаимодействия с устройством

Список команд, формат их использования и формат возвращаемых значений приведены в таблице Г1.

Таблица Г1 – Интерфейс взаимодействия с устройством

1	Команда	PassThruConnect
	Входной синтаксис	connect 5,c001 (ProtocolID, Flags)
	Выходной синтаксис	1 (ChannelID) / NOT_SUPPORTED (Error Code)
2	Команда	PassThruDisconnect
	Входной синтаксис	disconnect 1 (ChannelID)
	Выходной синтаксис	OK (Error Code)
3	Команда	PassThruReadMsgs
	Входной синтаксис	readmsgs 1,2,10 (ChannelID, NumMsgs, Timeout)
	Выходной синтаксис	Сообщения протокола, разделенные '\r' / ERROR (Error Code)
4	Команда	PassThruWriteMsgs
	Входной синтаксис	writemsgs 1,01A4C3D29D93F2A000C (ChannelID, Msg)
	Выходной синтаксис	OK (Error Code)
	Ограничения	Нет возможности отправить несколько сообщений за раз
5	Команда	PassThruStartPeriodicMsg
	Входной синтаксис	startperiodicmsgs 1,01A4C3D29D93F2A000C,100 (ChannelID, Msg, TimeInterval)
	Выходной синтаксис	3 (MsgID) / ERROR (Error Code)
6	Команда	PassThruStopPeriodicMsg
	Входной синтаксис	stopperiodicmsgs 1,3 (ChannelID, MsgID)
	Выходной синтаксис	OK (Error Code)

Продолжение таблицы Г1

7	Команда	PassThruStartMsgFilter
	Входной синтаксис	startmsgfilter 1,0,21df432ab432,2123afc321,21ad3424 (ChannelID, FilterType, MaskMsg, PatternMsg, FlowControlMsg)
	Выходной синтаксис	5 (MsgID) / ERROR (Error Code)
8	Команда	PassThruStopMsgFilter
	Входной синтаксис	stopmsgfilter 1,5 (ChannelID, MsgID)
	Выходной синтаксис	OK (Error Code)
9	Команда	PassThruSetProgrammingVoltage
	Входной синтаксис	setprogvoltage 6,20000 (PinNumber, Voltage)
	Выходной синтаксис	OK (Error Code)
10	Команда	PassThruReadVersion
	Входной синтаксис	readversion
	Выходной синтаксис	Строка версии программного обеспечения / ERROR (Error Code)
11	Команда	PassThruGetLastError
	Входной синтаксис	getlasterror
	Выходной синтаксис	Строка ошибки / ERROR (Error Code)
12	Команда	PassThruIoctl
	Входной синтаксис	ioctl 1,7,9032ad214f (ChannelID, IoctlID, Input)
	Выходной синтаксис	Выходная структура / ERROR (Error Code)
12.1	Команда	PassThruIoctl (GET_CONFIG)
	Входной синтаксис	ioctl 1,1,5,2,3,5,6,9 (количество и список запрашиваемых параметров)
	Выходной синтаксис	Список параметров, разделенные '\r' / ERROR (Error Code)
12.2	Команда	PassThruIoctl (SET_CONFIG)
	Входной синтаксис	ioctl 1,2,5,2,500,3,200,5,600,6,1,9,0 (количество пар, параметров и значений)
	Выходной синтаксис	OK (Error Code)

ПРИЛОЖЕНИЕ Д

Исходный код программы

main.h

```
/* Define to prevent recursive inclusion -------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------*/
#include "stm32f4xx_hal.h"

/* Private includes ----------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */
```

```
/* Private defines -----------------------------------------------
--------*/
#define USER_LED_Pin GPIO_PIN_13
#define USER_LED_GPIO_Port GPIOC
#define USER_BTN_Pin GPIO_PIN_0
#define USER_BTN_GPIO_Port GPIOA
#define CAN1_CS_Pin GPIO_PIN_1
#define CAN1_CS_GPIO_Port GPIOA
#define CAN2_CS_Pin GPIO_PIN_2
#define CAN2_CS_GPIO_Port GPIOA
#define CAN1_INT_Pin GPIO_PIN_0
#define CAN1_INT_GPIO_Port GPIOB
#define CAN1_INT_EXTI_IRQn EXTI0_IRQn
#define CAN2_INT_Pin GPIO_PIN_1
#define CAN2_INT_GPIO_Port GPIOB
#define CAN2_INT_EXTI_IRQn EXTI1_IRQn
#define ERROR_LED_Pin GPIO_PIN_7
#define ERROR_LED_GPIO_Port GPIOB
/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

main.c

```
/* USER CODE BEGIN Header */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------
--------*/
#include "main.h"
#include "usb_vcp/usb_device.h"

/* Private includes ----------------------------------------------------
--------*/
/* USER CODE BEGIN Includes */

#include "Button.h"
#include "PassThru/PassThruCore.h"

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------
--------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */
```

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
RTC_HandleTypeDef hrtc;

SPI_HandleTypeDef hspi1;

/* USER CODE BEGIN PV */

// User LED Variables
uint32_t ledTimer = 0;
uint32_t ledDuty = 512;
uint32_t ledPwmCounter = 0U;
uint32_t ledDutyTimer = 0;
uint32_t dutyDirection = 1;
uint32_t ledMode = 0;

Button_TypeDef user_btn;

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_RTC_Init(void);
static void MX_SPI1_Init(void);
/* USER CODE BEGIN PFP */

static void User_LED_Init();
static void breatheTick();

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */
```

```c
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_RTC_Init();
  MX_SPI1_Init();
  MX_USB_DEVICE_Init();
  /* USER CODE BEGIN 2 */

  // Initialize device modules
  PassThru_init(&hspi1);
  User_LED_Init();
  Btn_Init(&user_btn, USER_BTN_GPIO_Port, USER_BTN_Pin, GPIO_PIN_RESET);


  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if (Btn_Check(&user_btn)) {
      ledMode = !ledMode;
    }
```

```
    breatheTick();
    PassThru_tick();

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

  /** Initializes the RCC Oscillators according to the specified
parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSE|RCC_OSCILLATORTYPE_LSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 25;
  RCC_OscInitStruct.PLL.PLLN = 144;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 3;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
```

```
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) !=
HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief RTC Initialization Function
  * @param None
  * @retval None
  */
static void MX_RTC_Init(void)
{

  /* USER CODE BEGIN RTC_Init 0 */

  /* USER CODE END RTC_Init 0 */

  /* USER CODE BEGIN RTC_Init 1 */

  /* USER CODE END RTC_Init 1 */

  /** Initialize RTC Only
  */
  hrtc.Instance = RTC;
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
  hrtc.Init.AsynchPrediv = 127;
  hrtc.Init.SynchPrediv = 255;
  hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN RTC_Init 2 */

  /* USER CODE END RTC_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
```

```c
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(USER_LED_GPIO_Port, USER_LED_Pin, GPIO_PIN_SET);
```

```
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, CAN1_CS_Pin|CAN2_CS_Pin, GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(ERROR_LED_GPIO_Port, ERROR_LED_Pin, GPIO_PIN_SET);

  /*Configure GPIO pin : USER_LED_Pin */
  GPIO_InitStruct.Pin = USER_LED_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_OD;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(USER_LED_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : USER_BTN_Pin */
  GPIO_InitStruct.Pin = USER_BTN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(USER_BTN_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : CAN1_CS_Pin CAN2_CS_Pin */
  GPIO_InitStruct.Pin = CAN1_CS_Pin|CAN2_CS_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : CAN1_INT_Pin CAN2_INT_Pin */
  GPIO_InitStruct.Pin = CAN1_INT_Pin|CAN2_INT_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : ERROR_LED_Pin */
  GPIO_InitStruct.Pin = ERROR_LED_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_OD;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(ERROR_LED_GPIO_Port, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);

  HAL_NVIC_SetPriority(EXTI1_IRQn, 0, 0);

}

/* USER CODE BEGIN 4 */

static void User_LED_Init()
{
  ledTimer = HAL_GetTick();
```

```
  ledDutyTimer = HAL_GetTick();
}

static void breatheTick()
{
  if (ledMode == 0) {
    // имитация аппаратного счетчика
    ++ledPwmCounter;
    if (ledPwmCounter > 1023) {
      ledPwmCounter = 0;
    }

    // изменение коэффициента заполнения Ш?М раз в 1 мс
    if (ledDutyTimer + 1 < HAL_GetTick()) {
      ledDutyTimer = HAL_GetTick();
      if ((ledDuty == 0) || (ledDuty == 1023)) {
        dutyDirection = !dutyDirection;
      }

      ledDuty += (dutyDirection) ? 1 : -1;
    }

    // установка состояния выхода в зависимости от текущего состояния
счетчика и коэффициента заполнения
    HAL_GPIO_WritePin(USER_LED_GPIO_Port, USER_LED_Pin, (ledPwmCounter
< ledDuty) ? GPIO_PIN_RESET : GPIO_PIN_SET);
  } else {
    if (ledTimer + 500 < HAL_GetTick()){
      HAL_GPIO_TogglePin(USER_LED_GPIO_Port, USER_LED_Pin);
      ledTimer = HAL_GetTick();
    }
  }
}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error
return state */
  __disable_irq();

  HAL_GPIO_WritePin(ERROR_LED_GPIO_Port, ERROR_LED_Pin,
GPIO_PIN_RESET);
  while (1)
  {
```

```
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line
number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n",
file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

usbd_cdc_if.h

```
/* USER CODE BEGIN Header */
/* USER CODE END Header */

/* Define to prevent recursive inclusion ------------------------------
--------*/
#ifndef __USBD_CDC_IF_H__
#define __USBD_CDC_IF_H__

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------
--------*/
#include <usb_vcp/usblib/cdc/usbd_cdc.h>

/* USER CODE BEGIN INCLUDE */

/* USER CODE END INCLUDE */

/** @addtogroup STM32_USB_OTG_DEVICE_LIBRARY
  * @brief For Usb device.
  * @{
  */
```

```
/** @defgroup USBD_CDC_IF USBD_CDC_IF
  * @brief Usb VCP device module
  * @{
  */

/** @defgroup USBD_CDC_IF_Exported_Defines
USBD_CDC_IF_Exported_Defines
  * @brief Defines.
  * @{
  */
/* Define size for the receive and transmit buffer over CDC */
#define APP_RX_DATA_SIZE  1000
#define APP_TX_DATA_SIZE  1000
/* USER CODE BEGIN EXPORTED_DEFINES */

#define COMMAND_BUF_SIZE 1024
#define COMMAND_DELIMITER_SYMB '\r'

/* USER CODE END EXPORTED_DEFINES */

/**
  * @}
  */

/** @defgroup USBD_CDC_IF_Exported_Types USBD_CDC_IF_Exported_Types
  * @brief Types.
  * @{
  */

/* USER CODE BEGIN EXPORTED_TYPES */

typedef enum
{
  BUF_OK = 0U,
  BUF_FULL,
  BUF_NOTENOUGHSPACE,
  BUF_NOMSGAVAIL
} BufStatusTypeDef;

/* USER CODE END EXPORTED_TYPES */

/**
  * @}
  */

/** @defgroup USBD_CDC_IF_Exported_Macros USBD_CDC_IF_Exported_Macros
  * @brief Aliases.
  * @{
  */

/* USER CODE BEGIN EXPORTED_MACRO */
```

```
/* USER CODE END EXPORTED_MACRO */

/**
  * @}
  */

/** @defgroup USBD_CDC_IF_Exported_Variables
USBD_CDC_IF_Exported_Variables
  * @brief Public variables.
  * @{
  */

/** CDC Interface callback. */
extern USBD_CDC_ItfTypeDef USBD_Interface_fops_FS;

/* USER CODE BEGIN EXPORTED_VARIABLES */

/* USER CODE END EXPORTED_VARIABLES */

/**
  * @}
  */

/** @defgroup USBD_CDC_IF_Exported_FunctionsPrototype
USBD_CDC_IF_Exported_FunctionsPrototype
  * @brief Public functions declaration.
  * @{
  */

uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len);

/* USER CODE BEGIN EXPORTED_FUNCTIONS */

uint8_t Com_Msg_Available();
uint8_t Com_Read_Msg(uint8_t *buf, uint16_t len, uint16_t *msgLen);
void Com_Buf_Reset();


/* USER CODE END EXPORTED_FUNCTIONS */

#ifdef __cplusplus
}
#endif

#endif /* __USBD_CDC_IF_H__ */
```

usb_cdc_if.c

```
/* USER CODE BEGIN Header */
/* USER CODE END Header */
```

```
/* Includes ------------------------------------------------------------
--------*/
#include <usb_vcp/usbd_cdc_if.h>

/* USER CODE BEGIN INCLUDE */

/* USER CODE END INCLUDE */

/* Private typedef -----------------------------------------------------
--------*/
/* Private define ------------------------------------------------------
--------*/
/* Private macro -------------------------------------------------------
--------*/

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------
--------*/

/* USER CODE END PV */

/** @addtogroup STM32_USB_OTG_DEVICE_LIBRARY
  * @brief Usb device library.
  * @{
  */

/** @addtogroup USBD_CDC_IF
  * @{
  */

/** @defgroup USBD_CDC_IF_Private_TypesDefinitions
USBD_CDC_IF_Private_TypesDefinitions
  * @brief Private types.
  * @{
  */

/* USER CODE BEGIN PRIVATE_TYPES */

typedef struct {
      uint16_t firstChar;
      uint16_t lastChar;

      uint8_t noMessages;
      uint8_t empty;

      int16_t nextMessageEnd;

      uint8_t buf[COMMAND_BUF_SIZE];
} ComMsgBuf;

/* USER CODE END PRIVATE_TYPES */
```

```
/**
  * @}
  */

/** @defgroup USBD_CDC_IF_Private_Defines USBD_CDC_IF_Private_Defines
  * @brief Private defines.
  * @{
  */

/* USER CODE BEGIN PRIVATE_DEFINES */
/* USER CODE END PRIVATE_DEFINES */

/**
  * @}
  */

/** @defgroup USBD_CDC_IF_Private_Macros USBD_CDC_IF_Private_Macros
  * @brief Private macros.
  * @{
  */

/* USER CODE BEGIN PRIVATE_MACRO */

/* USER CODE END PRIVATE_MACRO */

/**
  * @}
  */

/** @defgroup USBD_CDC_IF_Private_Variables
USBD_CDC_IF_Private_Variables
  * @brief Private variables.
  * @{
  */
/* Create buffer for reception and transmission           */
/* It's up to user to redefine and/or remove those define */
/** Received data over USB are stored in this buffer      */
uint8_t UserRxBufferFS[APP_RX_DATA_SIZE];

/** Data to send over USB CDC are stored in this buffer   */
uint8_t UserTxBufferFS[APP_TX_DATA_SIZE];

/* USER CODE BEGIN PRIVATE_VARIABLES */

ComMsgBuf msgBuf =
{
  0,  // firstChar
  0,  // lastChar
  1,  // noMessages
  1,  // empty
```

```
  -1, // nextMessageEnd
  { '\0' } // buf
};

/* USER CODE END PRIVATE_VARIABLES */

/**
  * @}
  */

/** @defgroup USBD_CDC_IF_Exported_Variables
USBD_CDC_IF_Exported_Variables
  * @brief Public variables.
  * @{
  */

extern USBD_HandleTypeDef hUsbDeviceFS;

/* USER CODE BEGIN EXPORTED_VARIABLES */

/* USER CODE END EXPORTED_VARIABLES */

/**
  * @}
  */

/** @defgroup USBD_CDC_IF_Private_FunctionPrototypes
USBD_CDC_IF_Private_FunctionPrototypes
  * @brief Private functions declaration.
  * @{
  */

static int8_t CDC_Init_FS(void);
static int8_t CDC_DeInit_FS(void);
static int8_t CDC_Control_FS(uint8_t cmd, uint8_t* pbuf, uint16_t
length);
static int8_t CDC_Receive_FS(uint8_t* pbuf, uint32_t *Len);
static int8_t CDC_TransmitCplt_FS(uint8_t *pbuf, uint32_t *Len,
uint8_t epnum);

/* USER CODE BEGIN PRIVATE_FUNCTIONS_DECLARATION */

static uint8_t Com_Append_Data(uint8_t *buf, uint16_t len);
static uint16_t Get_Buffer_Free_Space();
static uint16_t Get_Cyclic_Distance(uint16_t start, uint16_t end,
uint16_t bufSize);

/* USER CODE END PRIVATE_FUNCTIONS_DECLARATION */

/**
  * @}
```

```
  */

USBD_CDC_ItfTypeDef USBD_Interface_fops_FS =
{
  CDC_Init_FS,
  CDC_DeInit_FS,
  CDC_Control_FS,
  CDC_Receive_FS,
  CDC_TransmitCplt_FS
};

/* Private functions ---------------------------------------------------*/
/**
  * @brief  Initializes the CDC media low layer over the FS USB IP
  * @retval USBD_OK if all operations are OK else USBD_FAIL
  */
static int8_t CDC_Init_FS(void)
{
  /* USER CODE BEGIN 3 */
  /* Set Application Buffers */
  USBD_CDC_SetTxBuffer(&hUsbDeviceFS, UserTxBufferFS, 0);
  USBD_CDC_SetRxBuffer(&hUsbDeviceFS, UserRxBufferFS);
  return (USBD_OK);
  /* USER CODE END 3 */
}

/**
  * @brief  DeInitializes the CDC media low layer
  * @retval USBD_OK if all operations are OK else USBD_FAIL
  */
static int8_t CDC_DeInit_FS(void)
{
  /* USER CODE BEGIN 4 */
  return (USBD_OK);
  /* USER CODE END 4 */
}

/**
  * @brief  Manage the CDC class requests
  * @param  cmd: Command code
  * @param  pbuf: Buffer containing command data (request parameters)
  * @param  length: Number of data to be sent (in bytes)
  * @retval Result of the operation: USBD_OK if all operations are OK
else USBD_FAIL
  */
static int8_t CDC_Control_FS(uint8_t cmd, uint8_t* pbuf, uint16_t
length)
{
  /* USER CODE BEGIN 5 */
  switch(cmd)
```

```
  {
    case CDC_SEND_ENCAPSULATED_COMMAND:

    break;

    case CDC_GET_ENCAPSULATED_RESPONSE:

    break;

    case CDC_SET_COMM_FEATURE:

    break;

    case CDC_GET_COMM_FEATURE:

    break;

    case CDC_CLEAR_COMM_FEATURE:

    break;

    case CDC_SET_LINE_CODING:

    break;

    case CDC_GET_LINE_CODING:

    break;

    case CDC_SET_CONTROL_LINE_STATE:

    break;

    case CDC_SEND_BREAK:

    break;

  default:
    break;
  }

  return (USBD_OK);
  /* USER CODE END 5 */
}

/**
  * @brief  Data received over USB OUT endpoint are sent over CDC
interface
  *         through this function.
  *
  *         @note
```

```
  *         This function will issue a NAK packet on any OUT packet
received on
  *         USB endpoint until exiting this function. If you exit this
function
  *         before transfer is complete on CDC interface (ie. using
DMA controller)
  *         it will result in receiving more data while previous ones
are still
  *         not sent.
  *
  * @param  Buf: Buffer of data to be received
  * @param  Len: Number of data received (in bytes)
  * @retval Result of the operation: USBD_OK if all operations are OK
else USBD_FAIL
  */
static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len)
{
  /* USER CODE BEGIN 6 */

  USBD_CDC_SetRxBuffer(&hUsbDeviceFS, &Buf[0]);
  USBD_CDC_ReceivePacket(&hUsbDeviceFS);

  if (Com_Append_Data(Buf, *Len) != BUF_OK) {
        Error_Handler();
  }

  return (USBD_OK);
  /* USER CODE END 6 */
}

/**
  * @brief  CDC_Transmit_FS
  *         Data to send over USB IN endpoint are sent over CDC
interface
  *         through this function.
  *         @note
  *
  *
  * @param  Buf: Buffer of data to be sent
  * @param  Len: Number of data to be sent (in bytes)
  * @retval USBD_OK if all operations are OK else USBD_FAIL or
USBD_BUSY
  */
uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len)
{
  uint8_t result = USBD_OK;
  /* USER CODE BEGIN 7 */
  USBD_CDC_HandleTypeDef *hcdc =
(USBD_CDC_HandleTypeDef*)hUsbDeviceFS.pClassData;
  if (hcdc->TxState != 0){
    return USBD_BUSY;
```

```
  }
  USBD_CDC_SetTxBuffer(&hUsbDeviceFS, Buf, Len);
  result = USBD_CDC_TransmitPacket(&hUsbDeviceFS);
  /* USER CODE END 7 */
  return result;
}

/**
  * @brief  CDC_TransmitCplt_FS
  *         Data transmitted callback
  *
  *         @note
  *         This function is IN transfer complete callback used to
inform user that
  *         the submitted Data is successfully sent over USB.
  *
  * @param  Buf: Buffer of data to be received
  * @param  Len: Number of data received (in bytes)
  * @retval Result of the operation: USBD_OK if all operations are OK
else USBD_FAIL
  */
static int8_t CDC_TransmitCplt_FS(uint8_t *Buf, uint32_t *Len, uint8_t
epnum)
{
  uint8_t result = USBD_OK;
  /* USER CODE BEGIN 13 */
  UNUSED(Buf);
  UNUSED(Len);
  UNUSED(epnum);
  /* USER CODE END 13 */
  return result;
}

/* USER CODE BEGIN PRIVATE_FUNCTIONS_IMPLEMENTATION */

static uint8_t Com_Append_Data(uint8_t *buf, uint16_t len)
{
  BufStatusTypeDef status = BUF_OK;

  if (Get_Buffer_Free_Space() < len) {
       return BUF_FULL;
  }

  if (msgBuf.lastChar + len >= COMMAND_BUF_SIZE) {
       uint16_t symbolsToCycle = COMMAND_BUF_SIZE - msgBuf.lastChar;
       strncpy((char*)&msgBuf.buf[msgBuf.lastChar], (char*)&buf[0],
symbolsToCycle);
       strncpy((char*)&msgBuf.buf[0], (char*)&buf[symbolsToCycle], len
- symbolsToCycle);

     msgBuf.lastChar = msgBuf.lastChar + len - COMMAND_BUF_SIZE;
```

```
  } else {
        strncpy((char*)&msgBuf.buf[msgBuf.lastChar], (char*)&buf[0],
len);

        msgBuf.lastChar = msgBuf.lastChar + len;
  }

  if (len > 0) {
        msgBuf.noMessages = 0;
        msgBuf.empty = 0;
  }

  return status;
}

static uint16_t Get_Buffer_Free_Space()
{
  uint16_t result;
  if (msgBuf.lastChar < msgBuf.firstChar && !msgBuf.empty) {
        result = msgBuf.firstChar - msgBuf.lastChar;
  } else {
        result = COMMAND_BUF_SIZE - (msgBuf.lastChar -
msgBuf.firstChar);
  }
  return result;
}

static uint16_t Get_Cyclic_Distance(uint16_t start, uint16_t end,
uint16_t bufSize)
{
  uint16_t result;
  if (end <= start) {
        result = COMMAND_BUF_SIZE - (start - end);
  } else {
        result = end - start;
  }
  return result;
}

uint8_t Com_Msg_Available()
{
  __disable_irq();
  uint8_t result = 0;
  uint16_t cur = msgBuf.firstChar;
  if (!msgBuf.empty && !msgBuf.noMessages && (msgBuf.nextMessageEnd ==
-1)) {
    do {
      if (msgBuf.buf[cur] == COMMAND_DELIMITER_SYMB) {
      result = 1;
      }
```

```
      ++cur;

      if (cur >= COMMAND_BUF_SIZE) {
      cur = 0;
      }
    }
        while ((cur != msgBuf.lastChar) && (result == 0));
  }

  if (!result) {
    msgBuf.noMessages = 1;
    msgBuf.nextMessageEnd = -1;
  } else {
    msgBuf.noMessages = 0;
    msgBuf.nextMessageEnd = cur;
  }

  __enable_irq();
  return result || msgBuf.nextMessageEnd != -1;
}

uint8_t Com_Read_Msg(uint8_t *buf, uint16_t bufLen, uint16_t *msgLen)
{
  BufStatusTypeDef result = BUF_OK;
  if (!Com_Msg_Available()) {
    return BUF_NOMSGAVAIL;
  }

  __disable_irq();
  uint16_t len = Get_Cyclic_Distance(msgBuf.firstChar,
msgBuf.nextMessageEnd, COMMAND_BUF_SIZE);

  if (bufLen < len) {
        return BUF_NOTENOUGHSPACE;
  }

  if (msgBuf.firstChar + len >= COMMAND_BUF_SIZE) {
    uint16_t symbolsToCycle = COMMAND_BUF_SIZE - msgBuf.firstChar;
    strncpy((char*)&buf[0], (char*)&msgBuf.buf[msgBuf.firstChar],
symbolsToCycle);
    strncpy((char*)&buf[symbolsToCycle], (char*)&msgBuf.buf[0], len -
symbolsToCycle);

    msgBuf.firstChar += len - COMMAND_BUF_SIZE;
  } else {
        strncpy((char*)&buf[0], (char*)&msgBuf.buf[msgBuf.firstChar],
len);

        msgBuf.firstChar += len;
  }
```

```
  msgBuf.nextMessageEnd = -1;

  if (msgBuf.firstChar == msgBuf.lastChar) {
        msgBuf.empty = 1;
        msgBuf.noMessages = 1;
  }

  *msgLen = len;

  __enable_irq();
  return result;
}

void Com_Buf_Reset()
{
      __disable_irq();
      msgBuf.empty = 1;
      msgBuf.firstChar = 0;
      msgBuf.lastChar = 0;
      msgBuf.noMessages = 1;
      msgBuf.nextMessageEnd = -1;
      __enable_irq();
}
```

VCPCommParser.h

```
#ifndef COMCOMMANDPARSER_H_
#define COMCOMMANDPARSER_H_


/* ------------------ VCP command parser includes ----------------- */

#include <PassThru/PassThruComm_if.h>

#include "usb_vcp/usbd_cdc_if.h"

/* ------------------ VCP command parser defines ----------------- */


/* ---------- VCP command parser public function declaration ---------
-- */

void VCP_getInterface(PassThruComm_ItfTypeDef *interface);

#endif /* COMCOMMANDPARSER_H_ */
```

VCPCommParser.c

```
/*
 * VCPCommParser.c
 *
 *  Created on: 23 дек. 2022 г.
```

```
 *      Author: mak22
 */

#include "usb_vcp/VCPCommParser.h"

/* ---------- VCP command parser private enum ----------- */

typedef enum {
  PARSE_ERROR = 0U,
  PARSE_COMMA,
  PARSE_END
} ParseStatus;

/* ---------- VCP command parser private variables ----------- */

#define COMMAND_COUNT 12
#define MAX_COMMAND_LEN 18


const char commands[COMMAND_COUNT][MAX_COMMAND_LEN] = {
    "connect",
    "disconnect",
    "readmsgs",
    "writemsgs",
    "startperiodicmsgs",
    "stopperiodicmsgs",
    "startmsgfilter",
    "stopmsgfilter",
    "setprogvoltage",
    "readversion",
    "getlasterror",
    "ioctl"
};

const char error_string[22][27] = {
    "STATUS_NOERROR\r",
    "ERR_NOT_SUPPORTED\r",
    "ERR_INVALID_CHANNEL_ID\r",
    "ERR_INVALID_PROTOCOL_ID\r",
    "ERR_NULLPARAMETER\r",
    "ERR_INVALID_IOCTL_VALUE\r",
    "ERR_INVALID_FLAGS\r",
    "ERR_FAILED\r",
    "ERR_DEVICE_NOT_CONNECTED\r",
    "ERR_TIMEOUT\r",
    "ERR_INVALID_MSG\r",
    "ERR_INVALID_TIME_INTERVAL\r",
    "ERR_EXCEEDED_LIMIT\r",
    "ERR_INVALID_MSG_ID\r",
    "ERR_INVALID_ERROR_ID\r",
    "ERR_INVAILD_IOCTL_ID\r",
```

```
    "ERR_BUFFER_EMPTY\r",
    "ERR_BUFFER_FULL\r",
    "ERR_BUFFER_OVERFLOW\r",
    "ERR_PIN_INVALID\r",
    "ERR_CHANNEL_IN_USE\r",
    "ERR_MSG_PROTOCOL_ID\r"
};

uint8_t question[] = { '?', COMMAND_DELIMITER_SYMB };


/* ---------- VCP command parser private function declaration --------
--- */

static uint8_t init(void);
static uint8_t deinit(void);
static uint8_t receiveCmd(uint8_t *cmd, PassThruParams *params);
static uint8_t sendAnswer(uint8_t *cmd, PassThruAnswer *ans);

static uint8_t parseInput(
    uint8_t *cmdBuf,
    uint8_t *cmd,
    PassThruParams *params
);

static uint8_t determineCommand(uint8_t *cmdBuf, uint8_t *cmd);
static uint8_t parseParams(uint8_t *cmdBuf, uint8_t cmd,
PassThruParams *params);

static uint8_t isCommand(uint8_t *cmdBuf, uint16_t cmdLen, uint8_t
*cmd);
static uint8_t parseMsg(uint8_t **iter, PassThruMessage *result);
static uint8_t parseInt(uint8_t **iter, uint32_t *result);
static uint8_t parseArray(uint8_t **iter, uint8_t *result);
static void prepareIntToSend(uint32_t num, uint8_t *buf);
static void prepareMsgToSend(PassThruMessage *msg, uint8_t *buf,
uint16_t *len);
static void sendErrorCode(uint32_t code);
static void sendVcpData(uint8_t *buf, uint16_t len);

void VCP_getInterface(PassThruComm_ItfTypeDef *interface)
{
  PassThruComm_ItfTypeDef _interface = {
    init,
    deinit,
    receiveCmd,
    sendAnswer
  };

  *interface = _interface;
  return;
```

```
}

static uint8_t init(void)
{
  uint8_t result = IF_OK;

  Com_Buf_Reset();

  return result;
}

static uint8_t deinit(void)
{
  uint8_t result = IF_OK;

  Com_Buf_Reset();

  return result;
}

static uint8_t receiveCmd(uint8_t *cmd, PassThruParams *params)
{
  uint8_t result = IF_OK;

  uint8_t cmdBuf[COMMAND_BUF_SIZE + 1];
  uint16_t msgLen = 0;
  switch (Com_Read_Msg(cmdBuf, COMMAND_BUF_SIZE, &msgLen)) {
    case BUF_NOMSGAVAIL:
      *cmd = NO_COMMAND;
      break;

    case BUF_OK:
      cmdBuf[msgLen] = '\0';
      if (parseInput(cmdBuf, cmd, params) != IF_OK) {
        sendVcpData(cmdBuf, msgLen);
        sendVcpData(question, sizeof(question));
      }

      break;

    case BUF_NOTENOUGHSPACE:
      result = IF_ERROR;
      Error_Handler();
      break;

    default:
      result = IF_ERROR;
      Error_Handler();
      break;
  }
```

```
  return result;
}

static uint8_t sendAnswer(uint8_t *cmd, PassThruAnswer *ans)
{
  uint8_t result = IF_OK;
  if (ans->errorCode != STATUS_NOERROR) {
    sendErrorCode(ans->errorCode);
    return result;
  }

  uint8_t sendBuf[200];
  uint16_t strLen = 0;

  switch (*cmd) {
    case CONNECT:
      prepareIntToSend(ans->Connect.channelId, sendBuf);
      sendVcpData(sendBuf, 8 + 1);
      break;

    case DISCONNECT:
      sendErrorCode(ans->errorCode);
      break;

    case READ_MSGS:
      prepareMsgToSend(&ans->ReadMsgs.msg, sendBuf, &strLen);
      sendVcpData(sendBuf, strLen);
      break;

    case WRITE_MSGS:

      break;

    case START_PERIODIC_MSG:

      break;

    case STOP_PERIODIC_MSG:

      break;

    case START_MSG_FILTER:

      break;

    case STOP_MSG_FILTER:

      break;

    case SET_PROGRAMMING_VOLTAGE:
```

```
      break;

    case READ_VERSION:

      break;

    case GET_LAST_ERROR:

      break;

    case IOCTL:

      break;

    default:

      break;
  }

  return result;
}

static uint8_t parseInput(
    uint8_t *cmdBuf,
    uint8_t *cmd,
    PassThruParams *params)
{
  uint8_t status = IF_OK;
  *cmd = NO_COMMAND;

  if (determineCommand(cmdBuf, cmd) != IF_OK ||
      parseParams(cmdBuf, *cmd, params) != IF_OK)
  {
    *cmd = NO_COMMAND;
    status = IF_ERROR;
    return status;
  }

  return status;
}

static uint8_t determineCommand(uint8_t *cmdBuf, uint8_t *cmd)
{
  uint8_t status = IF_OK;

  uint16_t cmdLen = 0;
  for (cmdLen = 0; (cmdBuf[cmdLen] != COMMAND_DELIMITER_SYMB) &&
         (cmdBuf[cmdLen] != ' '); ++cmdLen);

  if ((cmdBuf[cmdLen] == COMMAND_DELIMITER_SYMB) || isCommand(cmdBuf,
cmdLen, cmd) != IF_OK) {
```

```
    status = IF_ERROR;
  }

  return status;
}

static uint8_t parseParams(uint8_t *cmdBuf, uint8_t cmd,
PassThruParams *params)
{
  uint8_t status = IF_OK;
  uint8_t *iter = cmdBuf;
  for (; *(iter++) != ' '; );

  uint8_t parseOk = 0;
  switch (cmd) {
    case CONNECT:
      parseOk = (parseInt(&iter, &params->Connect.protocolId) ==
PARSE_COMMA) &&
        (parseInt(&iter, &params->Connect.flags) == PARSE_END);
      break;

    case DISCONNECT:
      parseOk = parseInt(&iter, &params->Disconnect.channelId) ==
PARSE_END;
      break;

    case READ_MSGS:
      parseOk = (parseInt(&iter, &params->ReadMsgs.channelId) ==
PARSE_COMMA) &&
        (parseInt(&iter, &params->ReadMsgs.numMsgs) == PARSE_COMMA) &&
        (parseInt(&iter, &params->ReadMsgs.timeout) == PARSE_END);
      break;

    case WRITE_MSGS:
      parseOk = (parseInt(&iter, &params->WriteMsgs.channelId) ==
PARSE_COMMA) &&
        (parseMsg(&iter, &params->WriteMsgs.msg) == PARSE_END);
      break;

    case START_PERIODIC_MSG:
      parseOk = (parseInt(&iter, &params->StartPeriodicMsg.channelId)
== PARSE_COMMA) &&
        (parseInt(&iter, &params->StartPeriodicMsg.interval) ==
PARSE_COMMA) &&
        (parseMsg(&iter, &params->StartPeriodicMsg.msg) == PARSE_END);
      break;

    case STOP_PERIODIC_MSG:
      parseOk = (parseInt(&iter, &params->StopPeriodicMsg.channelId)
== PARSE_COMMA) &&
```

```
        (parseInt(&iter, &params->StopPeriodicMsg.msgId) ==
PARSE_END);
      break;

    case START_MSG_FILTER:

      break;

    case STOP_MSG_FILTER:

      break;

    case SET_PROGRAMMING_VOLTAGE:

      break;

    case READ_VERSION:
      parseOk = 1;
      break;

    case GET_LAST_ERROR:
      parseOk = 1;
      break;

    case IOCTL:
      parseOk = (parseInt(&iter, &params->IOCTL.channelId) ==
PARSE_COMMA);

      uint8_t ioctlIdParseStatus;
      if (parseOk) {
        ioctlIdParseStatus = parseInt(&iter, &params->IOCTL.ioctlId);
      }
      parseOk = parseOk && ioctlIdParseStatus;

      if (parseOk) {
        switch (params->IOCTL.ioctlId) {
          case GET_CONFIG:
            if (ioctlIdParseStatus != PARSE_COMMA) {
              parseOk = 0;
              break;
            }

            parseOk = parseInt(&iter, &params-
>IOCTL.ioctl.sConfigList.numOfParams) == PARSE_COMMA;

            if (parseOk && params->IOCTL.ioctl.sConfigList.numOfParams
< MAX_CONFIG_PARAMS) {
              uint8_t i = 0;
              while (parseOk && (i < params-
>IOCTL.ioctl.sConfigList.numOfParams - 1)) {
```

```
                parseOk = parseInt(&iter, &params-
>IOCTL.ioctl.sConfigList.params[i++].parameter) == PARSE_COMMA;
              }
              if (parseOk) {
                parseOk = parseInt(&iter, &params-
>IOCTL.ioctl.sConfigList.params[i].parameter) == PARSE_END;
              }
            }
            break;

          case SET_CONFIG:
            if (ioctlIdParseStatus != PARSE_COMMA) {
              parseOk = 0;
              break;
            }

            parseOk = parseInt(&iter, &params-
>IOCTL.ioctl.sConfigList.numOfParams) == PARSE_COMMA;

            if (parseOk && params->IOCTL.ioctl.sConfigList.numOfParams
< MAX_CONFIG_PARAMS) {
              uint8_t i = 0;
              while (parseOk && (i < (params-
>IOCTL.ioctl.sConfigList.numOfParams - 1))) {
                parseOk = (parseInt(&iter, &params-
>IOCTL.ioctl.sConfigList.params[i].parameter) == PARSE_COMMA) &&
                          (parseInt(&iter, &params-
>IOCTL.ioctl.sConfigList.params[i++].value) == PARSE_COMMA);
              }
              if (parseOk) {
                parseOk = (parseInt(&iter, &params-
>IOCTL.ioctl.sConfigList.params[i].parameter) == PARSE_COMMA) &&
                          (parseInt(&iter, &params-
>IOCTL.ioctl.sConfigList.params[i].value) == PARSE_END);
              }
            }
            break;

          case READ_VBATT:
            if (ioctlIdParseStatus != PARSE_END) {
              parseOk = 0;
              break;
            }

            parseOk = 1;
            break;

          case FIVE_BAUD_INIT:
            parseOk = 1; // stub as not supported
            break;
```

```
case FAST_INIT:
  parseOk = 1; // stub as not supported
  break;

case CLEAR_TX_BUFFER:
  if (ioctlIdParseStatus != PARSE_END) {
    parseOk = 0;
    break;
  }

  parseOk = 1;
  break;

case CLEAR_RX_BUFFER:
  if (ioctlIdParseStatus != PARSE_END) {
    parseOk = 0;
    break;
  }

  parseOk = 1;
  break;

case CLEAR_PERIODIC_MSGS:
  if (ioctlIdParseStatus != PARSE_END) {
    parseOk = 0;
    break;
  }

  parseOk = 1;
  break;

case CLEAR_MSG_FILTERS:
  if (ioctlIdParseStatus != PARSE_END) {
    parseOk = 0;
    break;
  }

  parseOk = 1;
  break;

case CLEAR_FUNCT_MSG_LOOKUP_TABLE:
  parseOk = 1; // stub as not supported
  break;

case ADD_TO_FUNCT_MSG_LOOKUP_TABLE:
  parseOk = 1; // stub as not supported
  break;

case DELETE_FROM_FUNCT_MSG_LOOKUP_TABLE:
  parseOk = 1; // stub as not supported
  break;
```

```
          case READ_PROG_VOLTAGE:
            parseOk = 1; // stub as not supported
            break;

          default:
            Error_Handler();
            break;
        }
      }
      break;

    default:
      Error_Handler();
      break;
  }

  if (parseOk == 0) {
    status = IF_ERROR;
  }

  return status;
}

static uint8_t isCommand(uint8_t *cmdBuf, uint16_t cmdLen, uint8_t
*cmd)
{
  uint8_t status = IF_OK;
  uint16_t i = 0;
  cmdBuf[cmdLen] = '\0'; // it should be guaranteed that this symbols
is space

  uint8_t found = 1;
  while ((i < COMMAND_COUNT) && found != 0) {
    found = strcmp(commands[i++], (char*)cmdBuf);
  }

  if (found != 0) {
    status = IF_ERROR;
  }

  cmdBuf[cmdLen] = ' ';
  *cmd = i;

  return status;
}

static uint8_t parseMsg(uint8_t **iter, PassThruMessage *result)
{
  uint8_t status;
```

```
  status = (parseInt(iter, &result->ProtocolID) == PARSE_COMMA) &&
      (parseInt(iter, &result->RxStatus) == PARSE_COMMA) &&
      (parseInt(iter, &result->TxFlags) == PARSE_COMMA) &&
      (parseInt(iter, &result->Timestamp) == PARSE_COMMA) &&
      (parseInt(iter, &result->DataSize) == PARSE_COMMA) &&
      (parseInt(iter, &result->ExtraDataIndex) == PARSE_COMMA);

  if ((!status) || ((result->DataSize > 0) && (*((*iter) - 1) != ','))
||
      ((result->DataSize == 0) && (*((*iter) - 1) !=
COMMAND_DELIMITER_SYMB)))
  {
    // если парсинг не удался или ожидается содержимое, но встречена
не запятая,
    // или данных не ожидается, но
    status = PARSE_ERROR;
    return status;
  }

  if (result->DataSize != 0) {
    status = parseArray(iter, result->Data);
  }

  return status;
}

static uint8_t parseInt(uint8_t **iter, uint32_t *result)
{
  uint8_t status = 0;

  *result = 0;

  while ((**iter >= '0' && **iter <= '9') || (**iter >= 'a' && **iter
<= 'f')) {
    *result <<= 4;

    if (**iter >= '0' && **iter <= '9') {
      *result += **iter - '0';
    } else {
      *result += **iter - 'a' + 10;
    }

    ++(*iter);
  }

  switch (**iter) {
    case ',':
      status = PARSE_COMMA;
      break;

    case COMMAND_DELIMITER_SYMB:
```

```
      status = PARSE_END;
      break;

    default:
      status = PARSE_ERROR;
      break;
  }

  ++(*iter);

  return status;
}

static uint8_t parseArray(uint8_t **iter, uint8_t *result)
{
  uint8_t status = 0;
  uint8_t error = 0;
  uint16_t i = 0;

  while (((**iter >= '0' && **iter <= '9') || (**iter >= 'a' && **iter
<= 'f')) && !error) {
    uint8_t num = 0;

    if (**iter >= '0' && **iter <= '9') {
      num += **iter - '0';
    } else {
      num += **iter - 'a' + 10;
    }
    num <<= 4;

    ++(*iter);

    if (**iter >= '0' && **iter <= '9') {
      num += **iter - '0';
    } else if (**iter >= 'a' && **iter <= 'f') {
      num += **iter - 'a' + 10;
    } else {
      error = 1;
    }

    result[i] = num;
    ++(*iter);
  }

  if (!error) {
    switch (**iter) {
      case ',':
        status = PARSE_COMMA;
        break;

      case COMMAND_DELIMITER_SYMB:
```

```
        status = PARSE_END;
        break;

      default:
        status = PARSE_ERROR;
        break;
    }
  } else {
    status = PARSE_ERROR;
  }

  ++(*iter);

  return status;
}

static void prepareIntToSend(uint32_t num, uint8_t *buf)
{
  for (int i = 7; i >= 0; --i) {
    if ((num & 0x0FUL) <= 9) {
      buf[i] = '0' + (uint8_t)(num & 0x0FUL);
    } else {
      buf[i] = 'A' + (uint8_t)(num & 0x0FUL) - 10;
    }
    num >>= 4;
  }
  buf[8] = '\r';
}

static void prepareMsgToSend(PassThruMessage *msg, uint8_t *buf,
uint16_t *len)
{
  int i = 0;
  for (i = 0; i < msg->DataSize; ++i) {
    uint8_t num = msg->Data[i];

    if ((num & 0x0FUL) <= 9) {
      buf[i*2] = '0' + (uint8_t)(num & 0x0FUL);
    } else {
      buf[i*2] = 'A' + (uint8_t)(num & 0x0FUL) - 10;
    }

    num >>= 4;

    if ((num & 0x0FUL) <= 9) {
      buf[i*2+1] = '0' + (uint8_t)(num & 0x0FUL);
    } else {
      buf[i*2+1] = 'A' + (uint8_t)(num & 0x0FUL) - 10;
    }
  }
  buf[i*2] = '\r';
```

```
  buf[i*2+1] = '\0';
  *len = msg->DataSize * 2;
}

static void sendErrorCode(uint32_t code)
{
  uint8_t len = strlen(error_string[code]);
  sendVcpData(error_string[code], len);
}

static void sendVcpData(uint8_t *buf, uint16_t len)
{
  while (CDC_Transmit_FS(buf, len) != USBD_OK);
}
```

PassThru_if.h

```
/*
 * PassThru_if.h
 *
 *  Created on: 23 дек. 2022 г.
 *      Author: mak22
 */

#ifndef INC_PASSTHRU_PASSTHRU_DEF_H_
#define INC_PASSTHRU_PASSTHRU_DEF_H_

/* ---------------- PassThruInterface includes ------------------- */

#include "stdint.h"


#define MAX_CONFIG_PARAMS 15
#define MAX_BYTES_SARRAY MAX_CONFIG_PARAMS * 2 * sizeof(uint32_t)

/* ------------- PassThruInterface enums declaration -----------------
-- */

typedef enum {
  NO_COMMAND = 0U,
  CONNECT,
  DISCONNECT,
  READ_MSGS,
  WRITE_MSGS,
  START_PERIODIC_MSG,
  STOP_PERIODIC_MSG,
  START_MSG_FILTER,
  STOP_MSG_FILTER,
  SET_PROGRAMMING_VOLTAGE,
  READ_VERSION,
  GET_LAST_ERROR,
  IOCTL
```

```
} PassThruCommand;

typedef enum {
  GET_CONFIG = 0x1U,
  SET_CONFIG = 0x2U,
  READ_VBATT = 0x3U,
  FIVE_BAUD_INIT = 0x4U,
  FAST_INIT = 0x5U,
  CLEAR_TX_BUFFER = 0x7U,
  CLEAR_RX_BUFFER = 0x8U,
  CLEAR_PERIODIC_MSGS = 0x9U,
  CLEAR_MSG_FILTERS = 0xAU,
  CLEAR_FUNCT_MSG_LOOKUP_TABLE = 0xBU,
  ADD_TO_FUNCT_MSG_LOOKUP_TABLE = 0xCU,
  DELETE_FROM_FUNCT_MSG_LOOKUP_TABLE = 0xDU,
  READ_PROG_VOLTAGE = 0xEU
} PassThruIoctlId;

typedef enum {
  STATUS_NOERROR = 0U,
  ERR_NOT_SUPPORTED,
  ERR_INVALID_CHANNEL_ID,
  ERR_INVALID_PROTOCOL_ID,
  ERR_NULLPARAMETER,
  ERR_INVALID_IOCTL_VALUE,
  ERR_INVALID_FLAGS,
  ERR_FAILED,
  ERR_DEVICE_NOT_CONNECTED,
  ERR_TIMEOUT,
  ERR_INVALID_MSG,
  ERR_INVALID_TIME_INTERVAL,
  ERR_EXCEEDED_LIMIT,
  ERR_INVALID_MSG_ID,
  ERR_INVALID_ERROR_ID,
  ERR_INVAILD_IOCTL_ID,
  ERR_BUFFER_EMPTY,
  ERR_BUFFER_FULL,
  ERR_BUFFER_OVERFLOW,
  ERR_PIN_INVALID,
  ERR_CHANNEL_IN_USE,
  ERR_MSG_PROTOCOL_ID
} PassThruError;

typedef enum {
  J1850VPW = 1U,
  J1850PWM,
  ISO9141,
  ISO14230,
  CAN,
  ISO15765,
  SCI_A_ENGINE,
```

```
  SCI_A_TRANS,
  SCI_B_ENGINE,
  SCI_B_TRANS,
  MS_CAN = 0x10000U
} PassThruProtocolId;

/* ------------- PassThruInterface structs declaration ---------------
---- */

typedef struct {
  uint32_t parameter;
  uint32_t value;
} PassThruSConfig;

typedef struct {
  uint32_t ProtocolID;
  uint32_t RxStatus;
  uint32_t TxFlags;
  uint32_t Timestamp;
  uint32_t DataSize;
  uint32_t ExtraDataIndex;
  uint8_t Data[4128];
} PassThruMessage;

/// Shortened structure to reduce memory usage
typedef struct {
  uint32_t ProtocolID;
  uint32_t RxStatus;
  uint32_t TxFlags;
  uint32_t Timestamp;
  uint32_t DataSize;
  uint32_t ExtraDataIndex;
  uint8_t Data[64];
} PassThruMessageFilter;

typedef union {
  struct SCONFIG_LIST {
    uint32_t numOfParams;
    PassThruSConfig params[MAX_CONFIG_PARAMS * 2];
  } sConfigList;

  struct SBYTE_ARRAY {
    uint32_t numOfBytes;
    uint8_t bytes[MAX_BYTES_SARRAY];
  } sByteArray;

#ifdef EXTENDED_J2534_SUPPORT
  PassThruMessage msg;
#endif
} PassThruIoctl;
```

```
typedef union {
  struct {
    uint32_t protocolId;
    uint32_t flags;
  } Connect;

  struct  {
    uint32_t channelId;
  } Disconnect;

  struct  {
    uint32_t channelId;
    uint32_t numMsgs;
    uint32_t timeout;
  } ReadMsgs;

  struct {
    uint32_t channelId;
    uint32_t timeout;
    PassThruMessage msg;
  } WriteMsgs;

  struct {
    uint32_t channelId;
    uint32_t interval;
    PassThruMessage msg;
  } StartPeriodicMsg;

  struct {
    uint32_t channelId;
    uint32_t msgId;
  } StopPeriodicMsg;

  struct {
    uint32_t channelId;
    uint32_t filterType;
    PassThruMessageFilter mask;
    PassThruMessageFilter pattern;
    PassThruMessageFilter flowControl;
  } StartMsgFilter;

  struct {
    uint32_t channelId;
    uint32_t msgId;
  } StopMsgFilter;

  struct {
    uint32_t pinNumber;
    uint32_t voltage;
  } SetProgrammingVoltage;
```

```
  struct {

  } ReadVersion;

  struct {

  } GetLastError;

  struct {
    uint32_t channelId;
    uint32_t ioctlId;
    PassThruIoctl ioctl;
  } IOCTL;
} PassThruParams;

typedef struct {
  uint32_t errorCode;

  union {
    struct {
      uint32_t channelId;
    } Connect;

    struct {

    } Disconnect;

    struct {
      PassThruMessage msg;
    } ReadMsgs;

    struct {

    } WriteMsgs;

    struct {
      uint32_t msgId;
    } StartPeriodicMsg;

    struct {

    } StopPeriodicMsg;

    struct {
      uint32_t msgId;
    } StartMsgFilter;

    struct {

    } StopMsgFilter;
```

```
    struct {

    } SetProgrammingVoltage;

    struct {
      uint8_t *string;
    } ReadVersion;

    struct {
      uint8_t *string;
    } GetLastError;

    struct {
      uint32_t ioctlId;
      void *ptr;
    } IOCTL;
  };

} PassThruAnswer;

#endif /* INC_PASSTHRU_PASSTHRU_DEF_H_ */
```

PassThruComm_if.h

```
#ifndef INC_PASSTHRU_PASSTHRUCOMM_IF_H_
#define INC_PASSTHRU_PASSTHRUCOMM_IF_H_

#include "PassThru_def.h"

typedef struct {
  uint8_t (* Init)(void);
  uint8_t (* DeInit)(void);
  uint8_t (* ReceiveCmd)(uint8_t *cmd, PassThruParams *params);
  uint8_t (* SendAnswer)(uint8_t *cmd, PassThruAnswer *ans);
} PassThruComm_ItfTypeDef;

typedef enum {
  IF_OK = 0U,
  IF_ERROR
} PassThruCommError;

#endif /* INC_PASSTHRU_PASSTHRUCOMM_IF_H_ */
```

PassThruPeriph.h

```
/*
 * PassThruPeriph_if.h
 *
 *  Created on: Dec 27, 2022
 *      Author: mak22
 */
```

```
#ifndef INC_PASSTHRU_PASSTHRUPERIPH_IF_H_
#define INC_PASSTHRU_PASSTHRUPERIPH_IF_H_

#include "PassThru_def.h"

typedef struct {
  PassThruError (* Init)(void *this, void *params);
  PassThruError (* Connect)(void *this, PassThruParams *params);
  PassThruError (* Disconnect)(void *this);
  PassThruError (* ReadMsgs)(void *this, PassThruParams *params);
  PassThruError (* WriteMsgs)(void *this, PassThruParams *params);
  PassThruError (* SetFilter)(void *this, PassThruParams *params);
  PassThruError (* ResetFilter)(void *this, PassThruParams *params);
  PassThruError (* HandleIoctl)(void *this, PassThruParams *params);

  void (* interruptHandler)(void *this);

  uint8_t (* isConnected)(void *this);
  uint8_t (* isCapableOf)(void *this, PassThruProtocolId protocol);
} PassThruPeriph_ItfTypeDef;

#endif /* INC_PASSTHRU_PASSTHRUPERIPH_IF_H_ */
```

PassThruCore.h

```
/*
 * Control.h
 *
 *  Created on: 23 дек. 2022 г.
 *      Author: mak22
 */

#ifndef INC_PASSTHRU_PASSTHRUCORE_H_
#define INC_PASSTHRU_PASSTHRUCORE_H_

/* ------------------ PassThruCore includes ------------------- */

#include "PassThru/PassThru_def.h"
#include "PassThru/PassThruComm_if.h"

#include "stm32f4xx_hal.h"


/* ------------- PassThruCore struct declaration -------------------
*/

/* ------------- PassThruCore function declaration -------------------
*/


void PassThru_init(SPI_HandleTypeDef* _hspi);
void PassThru_tick(void);
```

```
#endif /* INC_PASSTHRU_PASSTHRUCORE_H_ */
```

PassThruCore.c

```
#include <PassThru/PassThruCore.h>
#include "PassThru/PassThruPeriph_if.h"
#include "usb_vcp/VCPCommParser.h"
#include "MCP2515/MCP2515.h"

/* -------------- PassThruCore private defines ----------------- */

#define PERIPH_COUNT 2

/* -------- PassThruCore private functions declarations ----------- */

static PassThruError ConnectHandler(PassThruParams *params,
PassThruAnswer *ans);
static PassThruError DisconnectHandler(PassThruParams *params,
PassThruAnswer *ans);
static PassThruError ReadMsgsHandler(PassThruParams *params,
PassThruAnswer *ans);
static PassThruError WriteMsgsHandler(PassThruParams *params,
PassThruAnswer *ans);
static PassThruError StartPeriodicMsgsHandler(PassThruParams *params,
PassThruAnswer *ans);
static PassThruError StopPeriodicMsgsHandler(PassThruParams *params,
PassThruAnswer *ans);
static PassThruError StartMsgFilterHandler(PassThruParams *params,
PassThruAnswer *ans);
static PassThruError StopMsgFilterHandler(PassThruParams *params,
PassThruAnswer *ans);
static PassThruError SetProgrammingVoltageHandler(PassThruParams
*params, PassThruAnswer *ans);
static PassThruError ReadVersionHandler(PassThruParams *params,
PassThruAnswer *ans);
static PassThruError GetLastErrorHandler(PassThruParams *params,
PassThruAnswer *ans);
static PassThruError IoctlHandler(PassThruParams *params,
PassThruAnswer *ans);


/* ------------- PassThruCore private variables ------------------ */

PassThruComm_ItfTypeDef comm_itf;

struct {
  void *periph;
  PassThruPeriph_ItfTypeDef itf;
} periphs[PERIPH_COUNT];
```

```
CAN_MCP2515_TypeDef Can1;
CAN_MCP2515_TypeDef Can2;

PassThruCommand command_id;
PassThruParams param_buf;
PassThruAnswer answer_buf;
uint8_t *last_error_string;

const uint8_t error_strings[1][1] = {
    ""
};

/* ------------- PassThru functions definition ------------------- */

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  switch (GPIO_Pin) {
    case CAN1_INT_Pin:
      periphs[0].itf.interruptHandler(periphs[0].periph);
      break;

    case CAN2_INT_Pin:
      periphs[1].itf.interruptHandler(periphs[1].periph);
      break;

    default:
      break;
  }
}

/*
 * Init PassThruCore, set communication interface
 * @retval None
 */
void PassThru_init(SPI_HandleTypeDef* _hspi)
{
  VCP_getInterface(&comm_itf);

  periphs[0].periph = &Can1;
  periphs[1].periph = &Can2;
  MCP2515_getInterface(&periphs[0].itf);
  MCP2515_getInterface(&periphs[1].itf);

  CAN_MCP2515_InitTypeDef can1_init = {
    _hspi,
    CAN1_CS_GPIO_Port,
    CAN1_CS_Pin
  };

  CAN_MCP2515_InitTypeDef can2_init = {
    _hspi,
```

```
    CAN2_CS_GPIO_Port,
    CAN2_CS_Pin
  };

  periphs[0].itf.Init(&Can1, &can1_init);
  periphs[1].itf.Init(&Can2, &can2_init);

  HAL_NVIC_EnableIRQ(CAN1_INT_EXTI_IRQn);
  HAL_NVIC_EnableIRQ(CAN2_INT_EXTI_IRQn);
}

/*
 * Serve PassThru events
 * @retval None
 */
void PassThru_tick(void)
{
  PassThruError error = STATUS_NOERROR;

  if(comm_itf.ReceiveCmd(&command_id, &param_buf) == IF_OK) {
    switch (command_id) {
      case NO_COMMAND:

        break;

      case CONNECT:
        error = ConnectHandler(&param_buf, &answer_buf);
        break;

      case DISCONNECT:
        error = DisconnectHandler(&param_buf, &answer_buf);
        break;

      case READ_MSGS:
        error = ReadMsgsHandler(&param_buf, &answer_buf);
        break;

      case WRITE_MSGS:
        error = WriteMsgsHandler(&param_buf, &answer_buf);
        break;

      case START_PERIODIC_MSG:
        error = StartPeriodicMsgsHandler(&param_buf, &answer_buf);
        break;

      case STOP_PERIODIC_MSG:
        error = StopPeriodicMsgsHandler(&param_buf, &answer_buf);
        break;

      case START_MSG_FILTER:
        error = StartMsgFilterHandler(&param_buf, &answer_buf);
```

```
        break;

      case STOP_MSG_FILTER:
        error = StopMsgFilterHandler(&param_buf, &answer_buf);
        break;

      case SET_PROGRAMMING_VOLTAGE:
        error = SetProgrammingVoltageHandler(&param_buf, &answer_buf);
        break;

      case READ_VERSION:
        error = ReadVersionHandler(&param_buf, &answer_buf);
        break;

      case GET_LAST_ERROR:
        error = GetLastErrorHandler(&param_buf, &answer_buf);
        break;

      case IOCTL:
        error = IoctlHandler(&param_buf, &answer_buf);
        break;

      default:
        Error_Handler();
        break;
    }
  }

  if (command_id != NO_COMMAND) {
    comm_itf.SendAnswer(&command_id, &answer_buf);
  }
}

static PassThruError ConnectHandler(PassThruParams *params,
PassThruAnswer *ans)
{
  PassThruError status = STATUS_NOERROR;

  switch (params->Connect.protocolId) {
    case CAN:
      ans->Connect.channelId = 0;
      status = periphs[0].itf.Connect(periphs[0].periph, params);
      break;

    case MS_CAN:
      ans->Connect.channelId = 1;
      status = periphs[1].itf.Connect(periphs[1].periph, params);
      break;

    default:
      status = ERR_NOT_SUPPORTED;
```

```
      break;
  }

  ans->errorCode = status;
  return status;
}

static PassThruError DisconnectHandler(PassThruParams *params,
PassThruAnswer *ans)
{
  if (params->Disconnect.channelId >= PERIPH_COUNT) {
    ans->errorCode = ERR_INVALID_CHANNEL_ID;
    return ERR_INVALID_CHANNEL_ID;
  }

  ans->errorCode = periphs[params->Disconnect.channelId].itf
      .Disconnect(periphs[params->Disconnect.channelId].periph);

  return ans->errorCode;
}

static PassThruError ReadMsgsHandler(PassThruParams *params,
PassThruAnswer *ans)
{

}

static PassThruError WriteMsgsHandler(PassThruParams *params,
PassThruAnswer *ans)
{

}

static PassThruError StartPeriodicMsgsHandler(PassThruParams *params,
PassThruAnswer *ans)
{

}

static PassThruError StopPeriodicMsgsHandler(PassThruParams *params,
PassThruAnswer *ans)
{

}

static PassThruError StartMsgFilterHandler(PassThruParams *params,
PassThruAnswer *ans)
{

}
```

```
static PassThruError StopMsgFilterHandler(PassThruParams *params,
PassThruAnswer *ans)
{

}

static PassThruError SetProgrammingVoltageHandler(PassThruParams
*params, PassThruAnswer *ans)
{

}

static PassThruError ReadVersionHandler(PassThruParams *params,
PassThruAnswer *ans)
{

}

static PassThruError GetLastErrorHandler(PassThruParams *params,
PassThruAnswer *ans)
{

}

static PassThruError IoctlHandler(PassThruParams *params,
PassThruAnswer *ans)
{

}
```

MCP2515.h

```
#ifndef INC_MCP2515_MCP2515_H_
#define INC_MCP2515_MCP2515_H_

/* ------------------ MCP2515 includes ------------------- */

#include "PassThru/PassThruPeriph_if.h"
#include "stm32f4xx_hal.h"


/* ----------- MCP2515 public struct declaration ------------ */

typedef struct {
  SPI_HandleTypeDef* hspi;
  GPIO_TypeDef* GPIO;
  uint16_t GPIO_CS_Pin;

  uint8_t connected;
} CAN_MCP2515_TypeDef;

typedef struct {
```

```
  SPI_HandleTypeDef* hspi;
  GPIO_TypeDef* GPIO;
  uint16_t GPIO_Pin;

} CAN_MCP2515_InitTypeDef;



/* ----------- MCP2515 public function declaration ------------ */

void MCP2515_getInterface(PassThruPeriph_ItfTypeDef *itf);

#endif /* INC_MCP2515_MCP2515_H_ */
```

MCP2515.c

```
#include "MCP2515/MCP2515.h"

/* ------------- MCP2515 define declaration -------------- */

/*
 * Register masks and bit offsets
 */

#define RXBNCTRL_RXM_MASK   (0x60U)
#define RXBNCTRL_RXM        (5)
#define RXBNCTRL_BUKT_MASK  (0x4U)
#define RXBNCTRL_BUKT       (2)

#define CNF3_SOF_MASK       (0x80U)
#define CNF3_SOF            (7)
#define CNF3_WAKFIL_MASK    (0x40U)
#define CNF3_WAKFIL         (6)
#define CNF3_PHSEG2_MASK    (0x7U)
#define CNF3_PHSEG2         (0)

#define CNF2_BLTMODE_MASK   (0x80U)
#define CNF2_BLTMODE        (7)
#define CNF2_SAM_MASK       (0x40U)
#define CNF2_SAM            (6)
#define CNF2_PHSEG1_MASK    (0x38U)
#define CNF2_PHSEG1         (3)
#define CNF2_PRSEG_MASK     (0x7U)
#define CNF2_PRSEG          (0)

#define CNF1_SJW_MASK       (0xC0U)
#define CNF1_SJW            (6)
#define CNF1_BRP_MASK       (0x3FU)
#define CNF1_BRP            (0)

#define CANCTRL_REQOP_MASK  (0xE0U)
#define CANCTRL_REQOP       (5)
```

```
#define CANSTAT_OPMOD_MASK  (0xE0U)
#define CANSTAT_OPMOD       (5)
#define CANSTAT_ICOD_MASK   (0x0EU)
#define CANSTAT_ICOD        (1)

/*
 * CAN speed timings
 */

#define MCP_TIMING_SJW      (0U)
#define MCP_TIMING_SAM      (0U)
#define MCP_TIMING_BTLMODE  (1U)

#define MCP_500KBPS_BRP     (0U)
#define MCP_500KBPS_PRSEG   (1U)
#define MCP_500KBPS_PHSEG1  (1U)
#define MCP_500KBPS_PHSEG2  (2U)
#define MCP_500KBPS_SAM     (0U)

#define MCP_250KBPS_BRP     (0U)
#define MCP_250KBPS_PRSEG   (4U)
#define MCP_250KBPS_PHSEG1  (4U)
#define MCP_250KBPS_PHSEG2  (4U)
#define MCP_250KBPS_SAM     (1U)

#define MCP_125KBPS_BRP     (1U)
#define MCP_125KBPS_PRSEG   (1U)
#define MCP_125KBPS_PHSEG1  (6U)
#define MCP_125KBPS_PHSEG2  (5U)
#define MCP_125KBPS_SAM     (1U)

/* ----------- MCP2515 private enum declaration ------------ */

typedef enum {
  MCP_OK = 0U,
  MCP_ERROR,
  MCP_INITERROR,
  MCP_TIMEOUT
} McpError;

typedef enum {
  MCP_RESET           = 0xC0,
  MCP_READ            = 0x03,
  MCP_READ_RX_BUFFER  = 0x90,
  MCP_WRITE           = 0x02,
  MCP_LOAD_TX_BUFFER  = 0x40,
  MCP_RTS             = 0x80,
  MCP_READ_STATUS     = 0xA0,
  MCP_RX_STATUS       = 0xB0,
  MCP_BIT_MODIFY      = 0x05
```

```
} McpCommand;

typedef enum {
  MCP_500KBPS,
  MCP_250KBPS,
  MCP_125KBPS
} McpSpeed;

typedef enum {
  MCP_NORMAL = 0U,
  MCP_SLEEP,
  MCP_LOOPBACK,
  MCP_LISTENONLY,
  MCP_CONFIGURATION
} McpMode;

typedef enum {
  MCP_RXF0SIDH = 0x00,
  MCP_RXF0SIDL = 0x01,
  MCP_RXF0EID8 = 0x02,
  MCP_RXF0EID0 = 0x03,
  MCP_RXF1SIDH = 0x04,
  MCP_RXF1SIDL = 0x05,
  MCP_RXF1EID8 = 0x06,
  MCP_RXF1EID0 = 0x07,
  MCP_RXF2SIDH = 0x08,
  MCP_RXF2SIDL = 0x09,
  MCP_RXF2EID8 = 0x0A,
  MCP_RXF2EID0 = 0x0B,
  MCP_BFPCTRL  = 0x0C,
  MCP_TXRTSCTRL= 0x0D,
  MCP_CANSTAT  = 0x0E,
  MCP_CANCTRL  = 0x0F,
  MCP_RXF3SIDH = 0x10,
  MCP_RXF3SIDL = 0x11,
  MCP_RXF3EID8 = 0x12,
  MCP_RXF3EID0 = 0x13,
  MCP_RXF4SIDH = 0x14,
  MCP_RXF4SIDL = 0x15,
  MCP_RXF4EID8 = 0x16,
  MCP_RXF4EID0 = 0x17,
  MCP_RXF5SIDH = 0x18,
  MCP_RXF5SIDL = 0x19,
  MCP_RXF5EID8 = 0x1A,
  MCP_RXF5EID0 = 0x1B,
  MCP_TEC      = 0x1C,
  MCP_REC      = 0x1D,
  MCP_RXM0SIDH = 0x20,
  MCP_RXM0SIDL = 0x21,
  MCP_RXM0EID8 = 0x22,
  MCP_RXM0EID0 = 0x23,
```

```
  MCP_RXM1SIDH = 0x24,
  MCP_RXM1SIDL = 0x25,
  MCP_RXM1EID8 = 0x26,
  MCP_RXM1EID0 = 0x27,
  MCP_CNF3     = 0x28,
  MCP_CNF2     = 0x29,
  MCP_CNF1     = 0x2A,
  MCP_CANINTE  = 0x2B,
  MCP_CANINTF  = 0x2C,
  MCP_EFLG     = 0x2D,
  MCP_TXB0CTRL = 0x30,
  MCP_TXB0SIDH = 0x31,
  MCP_TXB0SIDL = 0x32,
  MCP_TXB0EID8 = 0x33,
  MCP_TXB0EID0 = 0x34,
  MCP_TXB0DLC  = 0x35,
  MCP_TXB0DATA = 0x36,
  MCP_TXB1CTRL = 0x40,
  MCP_TXB1SIDH = 0x41,
  MCP_TXB1SIDL = 0x42,
  MCP_TXB1EID8 = 0x43,
  MCP_TXB1EID0 = 0x44,
  MCP_TXB1DLC  = 0x45,
  MCP_TXB1DATA = 0x46,
  MCP_TXB2CTRL = 0x50,
  MCP_TXB2SIDH = 0x51,
  MCP_TXB2SIDL = 0x52,
  MCP_TXB2EID8 = 0x53,
  MCP_TXB2EID0 = 0x54,
  MCP_TXB2DLC  = 0x55,
  MCP_TXB2DATA = 0x56,
  MCP_RXB0CTRL = 0x60,
  MCP_RXB0SIDH = 0x61,
  MCP_RXB0SIDL = 0x62,
  MCP_RXB0EID8 = 0x63,
  MCP_RXB0EID0 = 0x64,
  MCP_RXB0DLC  = 0x65,
  MCP_RXB0DATA = 0x66,
  MCP_RXB1CTRL = 0x70,
  MCP_RXB1SIDH = 0x71,
  MCP_RXB1SIDL = 0x72,
  MCP_RXB1EID8 = 0x73,
  MCP_RXB1EID0 = 0x74,
  MCP_RXB1DLC  = 0x75,
  MCP_RXB1DATA = 0x76
} McpRegister;


/* -------- MCP2515 private function declarations ----------- */

static PassThruError construct(void *_this, void *_params);
```

```
static PassThruError connect(void *_this, PassThruParams *params);
static PassThruError disconnect(void *_this);
static PassThruError sendMsg(void *_this, PassThruParams *params);
static PassThruError receiveMsg(void *_this, PassThruParams *params);
static PassThruError setFilter(void *_this, PassThruParams *params);
static PassThruError resetFilter(void *_this, PassThruParams *params);
static PassThruError handleIoctl(void *_this, PassThruParams *params);

static void interruptHandler(void *_this);

static uint8_t isConnected(void *this);
static uint8_t isCapableOf(void *this, PassThruProtocolId protocol);

static uint8_t init(CAN_MCP2515_TypeDef *this);
static void startSPI(CAN_MCP2515_TypeDef *this);
static void stopSPI(CAN_MCP2515_TypeDef *this);
static void reset(CAN_MCP2515_TypeDef *this);
static void readRegister(CAN_MCP2515_TypeDef *this, McpRegister reg,
uint8_t *buf);
static void writeRegister(CAN_MCP2515_TypeDef *this, McpRegister reg,
uint8_t val);
static void writeRegisters(CAN_MCP2515_TypeDef *this, McpRegister reg,
uint8_t *vals, uint8_t count);
static void bitSetRegister(CAN_MCP2515_TypeDef *this, McpRegister reg,
uint8_t mask, uint8_t val);
static void setSpeed(CAN_MCP2515_TypeDef *this, McpSpeed speed);
static McpMode getMode(CAN_MCP2515_TypeDef *this);
static McpMode setMode(CAN_MCP2515_TypeDef *this, McpMode mode);
static void prepareId(PassThruMessage *msg, uint8_t *buf);


/* ---------- MCP2515 public function definitions ----------- */

void MCP2515_getInterface(PassThruPeriph_ItfTypeDef *itf)
{
  PassThruPeriph_ItfTypeDef _itf = {
    construct,
    connect,
    disconnect,
    receiveMsg,
    sendMsg,
    setFilter,
    resetFilter,
    handleIoctl,
    interruptHandler,
    isConnected,
    isCapableOf
  };
  *itf = _itf;
}
```

```c
/* ---------- MCP2515 private function definitions ----------- */

static void startSPI(CAN_MCP2515_TypeDef *this)
{
  HAL_GPIO_WritePin(this->GPIO, this->GPIO_CS_Pin, GPIO_PIN_RESET);
}

static void stopSPI(CAN_MCP2515_TypeDef *this)
{
  HAL_GPIO_WritePin(this->GPIO, this->GPIO_CS_Pin, GPIO_PIN_SET);
}

static void reset(CAN_MCP2515_TypeDef *this)
{
  McpCommand cmd = MCP_RESET;
  startSPI(this);
  HAL_SPI_Transmit(this->hspi, &cmd, 1, 1);
  stopSPI(this);
}

static void readRegister(CAN_MCP2515_TypeDef *this, McpRegister reg,
uint8_t *buf)
{
  startSPI(this);
  uint8_t sendBuf[] = { MCP_READ, reg };
  HAL_SPI_Transmit(this->hspi, sendBuf, sizeof(sendBuf), 1);
  HAL_SPI_Receive(this->hspi, buf, 1, 1);
  stopSPI(this);
}

static void writeRegister(CAN_MCP2515_TypeDef *this, McpRegister reg,
uint8_t val)
{
  startSPI(this);
  uint8_t sendBuf[] = { MCP_WRITE, reg, val };
  HAL_SPI_Transmit(this->hspi, sendBuf, sizeof(sendBuf), 1);
  stopSPI(this);
}

static void writeRegisters(CAN_MCP2515_TypeDef *this, McpRegister reg,
uint8_t *vals, uint8_t count)
{
  startSPI(this);
  uint8_t sendBuf[] = { MCP_WRITE, reg };
  HAL_SPI_Transmit(this->hspi, sendBuf, sizeof(sendBuf), 1);
  HAL_SPI_Transmit(this->hspi, vals, count, 2);
  stopSPI(this);
}

static void bitSetRegister(CAN_MCP2515_TypeDef *this, McpRegister reg,
uint8_t mask, uint8_t val)
```

```
{
  startSPI(this);
  uint8_t sendBuf[] = { MCP_BIT_MODIFY, reg, mask, val };
  HAL_SPI_Transmit(this->hspi, sendBuf, sizeof(sendBuf), 1);
  stopSPI(this);
}

static void setSpeed(CAN_MCP2515_TypeDef *this, McpSpeed speed)
{
  uint8_t sendBuf[5] = {
    MCP_WRITE,
    MCP_CNF3,
  };

  switch (speed) {
    case MCP_500KBPS:
      sendBuf[2] = 0 << CNF3_SOF | 0 << CNF3_WAKFIL |
MCP_500KBPS_PHSEG2 << CNF3_PHSEG2;  // CNF3
      sendBuf[3] =
// CNF2
          1 << CNF2_BLTMODE |
          MCP_500KBPS_SAM << CNF2_SAM |
          MCP_500KBPS_PHSEG1 << CNF2_PHSEG1 |
          MCP_500KBPS_PRSEG << CNF2_PRSEG;
      sendBuf[4] = 0 << CNF1_SJW | MCP_500KBPS_BRP << CNF1_BRP;
// CNF1
      break;

    case MCP_250KBPS:
      sendBuf[2] = 0 << CNF3_SOF | 0 << CNF3_WAKFIL |
MCP_250KBPS_PHSEG2 << CNF3_PHSEG2;  // CNF3
      sendBuf[3] =
// CNF2
          1 << CNF2_BLTMODE |
          MCP_250KBPS_SAM << CNF2_SAM |
          MCP_250KBPS_PHSEG1 << CNF2_PHSEG1 |
          MCP_250KBPS_PRSEG << CNF2_PRSEG;
      sendBuf[4] = 0 << CNF1_SJW | MCP_250KBPS_BRP << CNF1_BRP;
// CNF1
      break;

    case MCP_125KBPS:
      sendBuf[2] = 0 << CNF3_SOF | 0 << CNF3_WAKFIL |
MCP_125KBPS_PHSEG2 << CNF3_PHSEG2;  // CNF3
      sendBuf[3] =
// CNF2
          1 << CNF2_BLTMODE |
          MCP_125KBPS_SAM << CNF2_SAM |
          MCP_125KBPS_PHSEG1 << CNF2_PHSEG1 |
          MCP_125KBPS_PRSEG << CNF2_PRSEG;
```

```
      sendBuf[4] = 0 << CNF1_SJW | MCP_125KBPS_BRP << CNF1_BRP;
// CNF1
      break;

    default:
      Error_Handler();
      break;
  }

  startSPI(this);
  HAL_SPI_Transmit(this->hspi, sendBuf, sizeof(sendBuf), 1);
  stopSPI(this);
}

static uint8_t init(CAN_MCP2515_TypeDef *this)
{
  // Controller reset
  stopSPI(this);
  reset(this);
  HAL_Delay(2);

  // Check if there is MCP2515 connected
  uint8_t readBuf[1];
  readRegister(this, MCP_CANCTRL, readBuf);

  if (readBuf[0] != 0x87) {
    Error_Handler();
    return MCP_INITERROR;
  }

  // Allow to receive all messages, regardless filters, and rollover
  bitSetRegister(this, MCP_RXB0CTRL, RXBNCTRL_RXM_MASK |
RXBNCTRL_BUKT_MASK,
      (0b11 << RXBNCTRL_RXM) | (1 << RXBNCTRL_BUKT));
  bitSetRegister(this, MCP_RXB1CTRL, RXBNCTRL_RXM_MASK, 0b11 <<
RXBNCTRL_RXM);

  setSpeed(this, MCP_500KBPS);

  return MCP_OK;
}

static McpMode getMode(CAN_MCP2515_TypeDef *this)
{
  uint8_t result;
  readRegister(this, MCP_CANSTAT, &result);
  result = (result & CANSTAT_OPMOD_MASK) >> CANSTAT_OPMOD;

  switch (result) {
    case 0x0:
      return MCP_NORMAL;
```

```
    case 0x1:
      return MCP_SLEEP;

    case 0x2:
      return MCP_LOOPBACK;

    case 0x3:
      return MCP_LISTENONLY;

    case 0x4:
      return MCP_CONFIGURATION;

    default:
      Error_Handler();
      break;
  }
  return -1;
}

static McpMode setMode(CAN_MCP2515_TypeDef *this, McpMode mode)
{
  bitSetRegister(this, MCP_CANCTRL, CANCTRL_REQOP_MASK, mode <<
CANCTRL_REQOP);

  return mode;
}

static void interruptHandler(void *_this)
{
  CAN_MCP2515_TypeDef *this = _this;

  uint8_t result[1];
  readRegister(this, MCP_CANINTF, result);
}

static PassThruError construct(void *_this, void *_params)
{
  CAN_MCP2515_TypeDef *this = _this;
  CAN_MCP2515_InitTypeDef *params = _params;

  this->hspi = params->hspi;
  this->GPIO = params->GPIO;
  this->GPIO_CS_Pin = params->GPIO_Pin;

  this->connected = 0;

  return (init(this) == MCP_OK) ? STATUS_NOERROR : ERR_FAILED;
}

static PassThruError connect(void *_this, PassThruParams *params)
```

```
{
  CAN_MCP2515_TypeDef *this = _this;

  if (this->connected) {
    return ERR_CHANNEL_IN_USE;
  }

  if (params->Connect.flags > 0) {
    return ERR_NOT_SUPPORTED;
  }

  init(this);

  setMode(this, MCP_NORMAL);

  this->connected = 1;
  return STATUS_NOERROR;
}

static PassThruError disconnect(void *_this)
{
  CAN_MCP2515_TypeDef *this = _this;

  setMode(this, MCP_CONFIGURATION);

  this->connected = 0;
  return STATUS_NOERROR;
}

static PassThruError sendMsg(void *_this, PassThruParams *params)
{

}

static PassThruError receiveMsg(void *_this, PassThruParams *params)
{

}

static PassThruError setFilter(void *_this, PassThruParams *params)
{

}

static PassThruError resetFilter(void *_this, PassThruParams *params)
{

}

static PassThruError handleIoctl(void *_this, PassThruParams *params)
{
```

```
}

static uint8_t isConnected(void *this)
{

}

static uint8_t isCapableOf(void *this, PassThruProtocolId protocol)
{
  return 0;
}

static void prepareId(PassThruMessage *msg, uint8_t *buf)
{

}
```