МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРЕОБРАЗОВАТЕЛЬ ЛОГИЧЕСКИХ СИГНАЛОВ МЕЖДУ ШИНАМИ SPI И CAN

Студент 09.06.2026 Перфилов А.В.
(подпись, дата)

Руководитель 10.06.2026 Егоров В.Ю.
(подпись, дата)

Нормоконтролёр 10.06.2026 А.С. Бычков
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от 22.05.2026 г. №14)

Заведующий кафедрой _____ М.А. Митрохин
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 19 июня 2026 г. № 41)

Секретарь ГЭК _____ Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«07» 04 2026г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Перфилов Александр Валерьевич гр.22ВВП1 факультета вычислительной техники.

2.Тема работы: «Преобразователь логических сигналов между шинами spi и can»

Тема утверждена приказом ПГУ №347/о от "7" апреля 2026 г.

3. Руководитель бакалаврской работы к.т.н., доцент, каф. ВТ ПГУ Егоров В.Ю.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать узел контроллера CAN, выполняющего функции преобразователя между шинами интерфейсов SPI и CAN, а также маскирования шины SPI. Работа включается в себя разработку принципиальной схемы и программного обеспечения изделия в его заданной части.

5. Перечень подлежащих разработке вопросов

Требуется провести анализ существующих микросхем, выполняющих преобразование интерфейсов SPI и CAN, выявить их достоинства и недостатки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Ознакомление с темой, разработка плана проведения работ	4	07.04.2026	
2	Анализ требований	20	07.04.2026 – 10.04.2026	
3	Изучение документации	20	11.04.2026 – 14.04.2026	
4	Доработка принципиальной схемы	60	15.04.2026 – 19.04.2026	
5	Написание программы	60	20.04.2026 – 02.04.2026	
6	Отладка программы	20	03.05.2026– 15.05.2026	
7	Оформление пояснительной записки	20	16.05.2026 – 21.05.2026	

Дата выдачи задания "7" апреля 2026 г.

Руководитель бакалаврской работы _____ Егоров В.Ю.
(подпись, дата)

Задание к исполнению принял студент _____ Перфилов А.В.
(подпись, дата)

Работу к защите допустить
Декан факультета 17.06.2020 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 121 лист, 43 рисунка 6 таблиц, 8 источников, 4 приложения.

ПРЕОБРАЗОВАТЕЛЬ, ИНТЕРФЕЙС, РЕГИСТР, КОНТРОЛЛЕР, УЗЕЛ, МАСКИРОВАНИЕ, CAN, SPI

Цель работы – Разработать узел контроллера CAN, и написать к нему программное обеспечение. Узел должен являться преобразователем между интерфейсами SPI и CAN, а также маскировать информацию.

В процессе работы был проведен анализ существующих микросхем, разработана принципиальная схема, разработано программное обеспечение.

Разработка проводилась в среде программирования IAR Embedded Workbench. Для тестирования использовался кабель с преобразователем USB-CAN, а также программное обеспечение к нему для отправки пакетов.

В результате проведенной работы был разработан узел, удовлетворяющий поставленной задаче.

Данный узел успешно внедрен в аппаратуру.

Содержание

Перечень сокращений и обозначений 6
Введение 7
1 Преобразователь логических сигналов между шинами spi и can 8
1.1. Постановка задачи 8
1.2 Анализ существующих аналогов 9
1.3 Выводы 13
2 Структура аппаратно-программного комплекса 15
2.1 Общие сведения 15
2.2 Аппаратная часть 20
2.3 Программная часть 22
3 Тестирование 37
3.1 Тест 1 38
3.2 Тест 2 42
4 Охрана труда 43
4.1 Вред здоровью при работе с ПЭВМ 46
4.2 Требования безопасности при использовании устройства, коммутирующего сетевое напряжение 220В 47
4.3 Требования условий эксплуатации устройства управления электроприборами на основе микроконтроллера STM32 48
Заключение 49
Список использованных источников 50
Приложение А – UML Диаграммы 52
Приложение Б – Схема электрическая принципиальная 55
Приложение В – Листинг программы 58
Приложение Г – Презентация 116

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

КИ – Контроллер индикации

ПО – Программное обеспечение

ПЭВМ – Персональная электронно-вычислительная машина

СБ – Спецблок

CAN (Controller Area Network) –промышленная сеть контроллеров

I2C (Inter-Integrated Circuit) — Последовательная асимметричная шина

SPI (Serial Peripheral Interface) – Последовательный периферийный интерфейс

UML (Unified Modeling Language) – унифицирован язык моделирования.

USART (Universal Synchronous-Asynchronous Receiver-Transmitter) — Универсальный синхронно-асинхронный приёмопередатчик

Введение

В современной электронике активно используется передача цифрового сигнала между контроллерами, датчиками и другими компонентами. С их помощью можно организовать различной сложности промышленные системы. Для взаимодействия между элементами системы применяются интерфейсы обмена. Развитие современных электронных систем приводит к тому, что в одном устройстве всё чаще приходится сочетать несколько интерфейсов, каждый из которых выполняет свою роль в общей архитектуре. Встраиваемые системы становятся более сложными, увеличивается количество узлов, растут требования к скорости обмена, устойчивости к помехам и безопасности передаваемых данных. В таких условиях особое значение приобретает правильная организация взаимодействия между компонентами, использующими разные протоколы передачи. Кроме того, современные системы предъявляют повышенные требования к надёжности обмена информацией. Ошибки в передаче данных, некорректные команды или нарушение временных характеристик могут приводить к сбоям в работе оборудования, что делает задачу согласования интерфейсов особенно актуальной. В ряде случаев требуется не только преобразование формата данных, но и обеспечение дополнительной защиты, фильтрации или контроля сообщений, поступающих от внешних узлов.

В связи с этим возрастает необходимость разработки специализированных узлов, способных выполнять функции промежуточного звена между интерфейсами и обеспечивать безопасный, стабильный и предсказуемый обмен данными. Такие модули позволяют адаптировать устройство к условиям эксплуатации, повысить его устойчивость к внешним воздействиям и обеспечить корректную работу всей системы в целом.

1 Преобразователь логических сигналов между шинами spi и can

1.1 Постановка задачи

На предприятии, где выполнялась данная работа, была поставлена задача, в виде доработки платы индикации путем интегрирования преобразователя логических данных. Данная необходимость возникла после переоценки текущей архитектуры передачи данных в микроконтроллер, она не обеспечивала должной изоляции от внешних управляющих команд. Для повышения надежности системы было принято решение добавить аппаратный промежуточный модуль, который будет маскировать принимаемые сообщения. Также после доработки аппаратной части, нужно дополнить программное обеспечение для взаимодействия с новой микросхемой.

Для того чтобы обеспечить маскирование нужно было выбрать подходящий протокол передачи, который будет использоваться между микроконтроллером и трансивером CAN интерфейса. Выбранный интерфейс должен быть максимально быстрым и простым в организации линии передачи, так как ее в дальнейшем нужно будет маскировать.

USART — асинхронный последовательный интерфейс, широко используемый для обмена между микроконтроллерами и периферией. Передача осуществляется в виде последовательности байтов с использованием старт и стоп-битов и опционального бита чётности. В данном случае служебные биты не позволяют правильно замаскировать сообщение не затронув их, без использования других контроллеров.

I²C — двухпроводной синхронный интерфейс с адресацией, предназначенный для связи множества низкоскоростных устройств на одной плате или в пределах одного узла. Он экономит проводку и удобен для подключения датчиков и конфигурационных EEPROM. Данный интерфейс так же не подходит из-за низкой скорости передачи и большого числа служебных битов.

SPI — синхронный интерфейс, обеспечивающий полно-дуплексную передачу данных и высокую пропускную способность на коротких

расстояниях. Он прост в аппаратной реализации, не имеет служебных битов, так как их функцию берут на себя линии CS и CLK. Таким образом данный интерфейс полностью удовлетворяет требованиям промежуточного интерфейса.

Таким образом задача состояла в том, чтобы добавить узел SPI-CAN преобразователя, а также доработки программного обеспечения в части взаимодействия с новым узлом платы.

1.2 Анализ существующих аналогов

В рамках анализа существующих аналогов были рассмотрены решения, представленные в виде микросхем – преобразователей протоколов spi и can. Проведенный анализ существующих предложений показал, что на рынке нет доступных микросхем, сочетающих в себе маскирующие свойства. В данном сегменте преобладают 2 производителя, microchip с линейкой MCP и Texas Instruments с с линейкой TCAN.

Микросхема MCP2517FD представляет собой CAN-контроллер с интерфейсом SPI. Взаимодействие с управляющим устройством осуществляется по интерфейсу SPI. Предусматривает работу в двух режимах работы, в режиме CAN 2.0 и CAN FD. Имеет программируемые выходы, которые могут быть как GPIO на вход или на выход, так и могут быть использованы для организации взаимодействия через прерывания. Для хранения сообщений существует оперативная память размером 2 КБ, которая поделена на 31 очередь, каждая из которых может быть сконфигурирована на хранение передаваемых или принимаемых сообщений. Существуют специальные очереди, например отдельная очередь для отправляемых сообщений и очередь для событий передачи, которая хранит информацию о передаваемых пакетах, такую как идентификатор сообщений, длина данных, флаги, временную метку и индекс буфера, с которым связано это сообщение. Для фильтрации сообщений используются 32 фильтра и маски. Их можно связать с конкретной очередью. Фильтрация осуществляется в зависимости от id сообщения. Предусмотрена работа как со стандартным id, так и с расширенным. Для защиты данных используется код коррекции ошибок для

оперативной памяти, хранящей сообщения. Так же для сообщений передаваемых через spi интерфейс предусмотрена контрольная сумма (CRC) для обнаружения шума на шине. Данная микросхема имеет перечень своих регистров, обеспечивающих управление микросхемой. Для подключения к физической линии связи требуется внешний CAN-трансивер, обеспечивающий согласование электрических уровней сигнала. Для работы микросхемы можно использовать внутренние резонаторы на 40 Мг, 20 Мг или 4 Мг или внешнее тактирование [1]. Структура микросхемы представлена на рисунке 1.

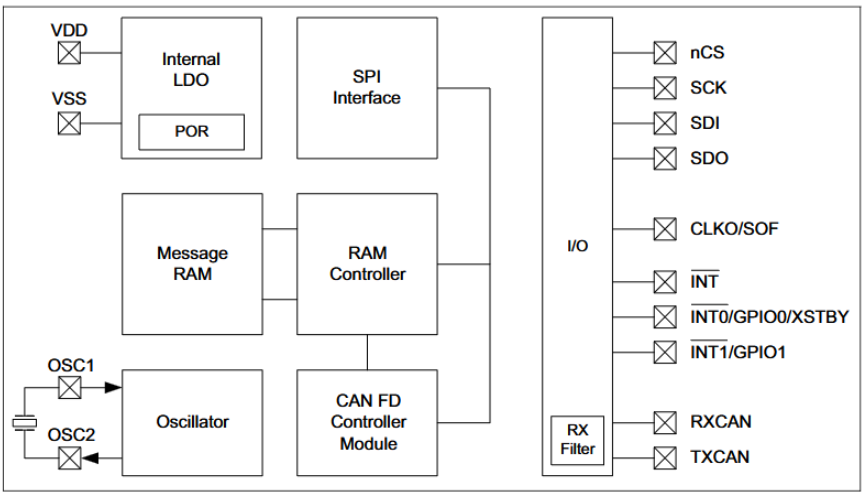


Рисунок 1 – Блок-диаграмма MCP2517fd

К основным недостаткам контроллера MCP2517FD можно отнести сравнительно сложную процедуру инициализации и настройки;

Микросхема MCP2515 представляет собой CAN-контроллер, работающий по интерфейсу SPI и предназначенный для организации взаимодействия микроконтроллера с шиной CAN 2.0B, поддерживает как стандартные, так и расширенные идентификаторы сообщений. Внутри микросхемы реализованы буферы для приема и передачи сообщений, включая два приемных буфера и три передающих буфера, которые могут быть настроены с разным приоритетом отправки. Для фильтрации входящих сообщений предусмотрены маски и фильтры, позволяющие отбирать кадры по идентификатору и направлять их в соответствующие буферы. Взаимодействие с управляющим устройством осуществляется через SPI с использованием набора внутренних регистров, отвечающих за конфигурацию режимов работы,

управление передачей и приемом, а также обработку ошибок. Микросхема поддерживает механизм прерываний для уведомления микроконтроллера о событиях, таких как прием сообщения, завершение передачи или возникновение ошибок на шине, для синхронизации работы используется внешний кварцевый резонатор, также предусмотрены встроенные средства контроля ошибок CAN, включая обнаружение битовых, формата и контрольная сумма (CRC). Структуру микросхемы представлена на рисунке 2.

К основным недостаткам контроллера MCP2515 можно отнести:

- отсутствие дополнительной защиты памяти;
- маленький объем буферов и отсутствие оперативной памяти для хранения сообщений;
- отсутствие возможности работы с CAN FD.

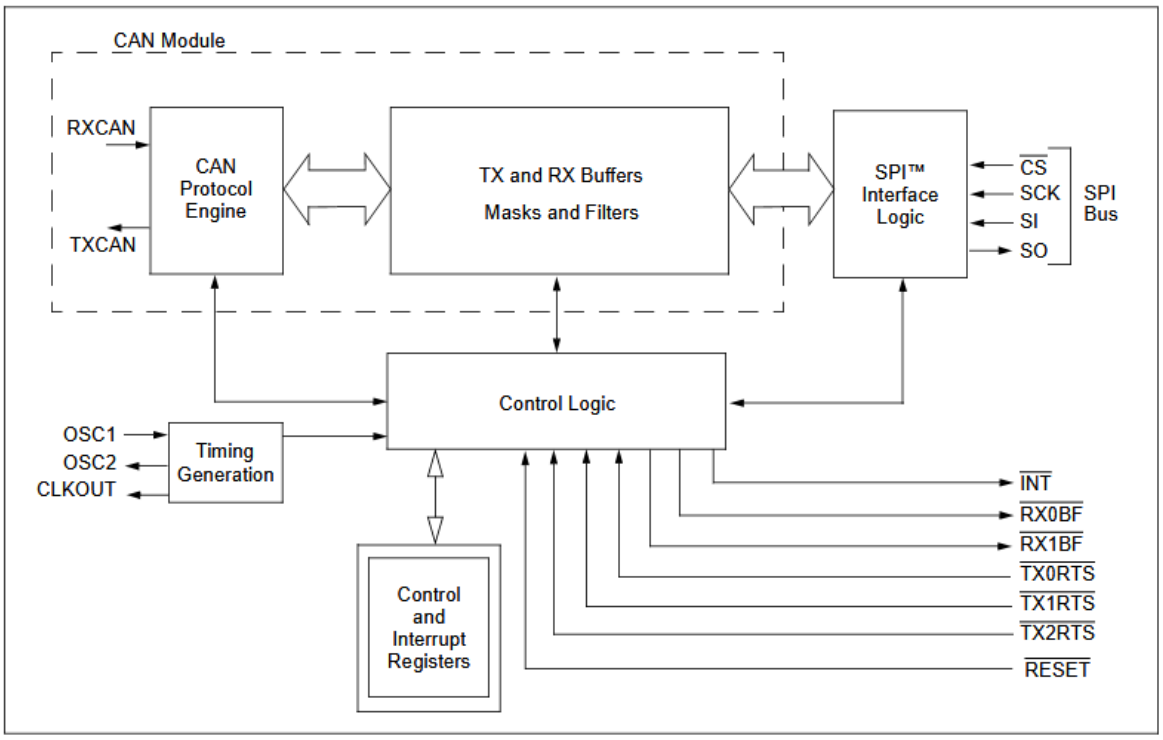


Рисунок 2 – Блок-диаграмма MCP2515

Микросхема TCAN4550 представляет собой устройство, сочетающее в себе как сам контроллер, так и CAN-трансивер, благодаря чему не требуется использование внешнего преобразователя уровней. Взаимодействие с управляющим устройством осуществляется по SPI с использованием внутренних регистров и области памяти сообщений. Архитектура контроллера

предусматривает использование оперативной памяти для хранения кадров, организованной в виде настраиваемых буферов и очередей для приема и передачи сообщений. Поддерживаются как стандартные, так и расширенные идентификаторы, а также работа в классическом режиме CAN 2.0 и в режиме CAN FD. Для фильтрации входящих сообщений используются настраиваемые фильтры и маски, позволяющие направлять сообщения в заданные буферы, предусмотрен механизм временных меток и хранения служебной информации о передаче и приеме кадров. Микросхема поддерживает систему прерываний для уведомления микроконтроллера о событиях, включая прием сообщений, завершение передачи и ошибки на шине. Реализованы встроенные механизмы диагностики и обнаружения ошибок протокола CAN, а также защита передачи данных по SPI с использованием контрольных сумм, дополнительно устройство содержит средства управления питанием и режимы пониженного энергопотребления, а также встроенный сторожевой таймер. Кроме того, в микросхеме реализован встроенный стабилизатор напряжения LDO, обеспечивающий формирование внутреннего питающего напряжения для логики и позволяющий упростить внешнюю схему питания за счет уменьшения количества дополнительных компонентов. Структура микросхемы представлена на рисунке 3.

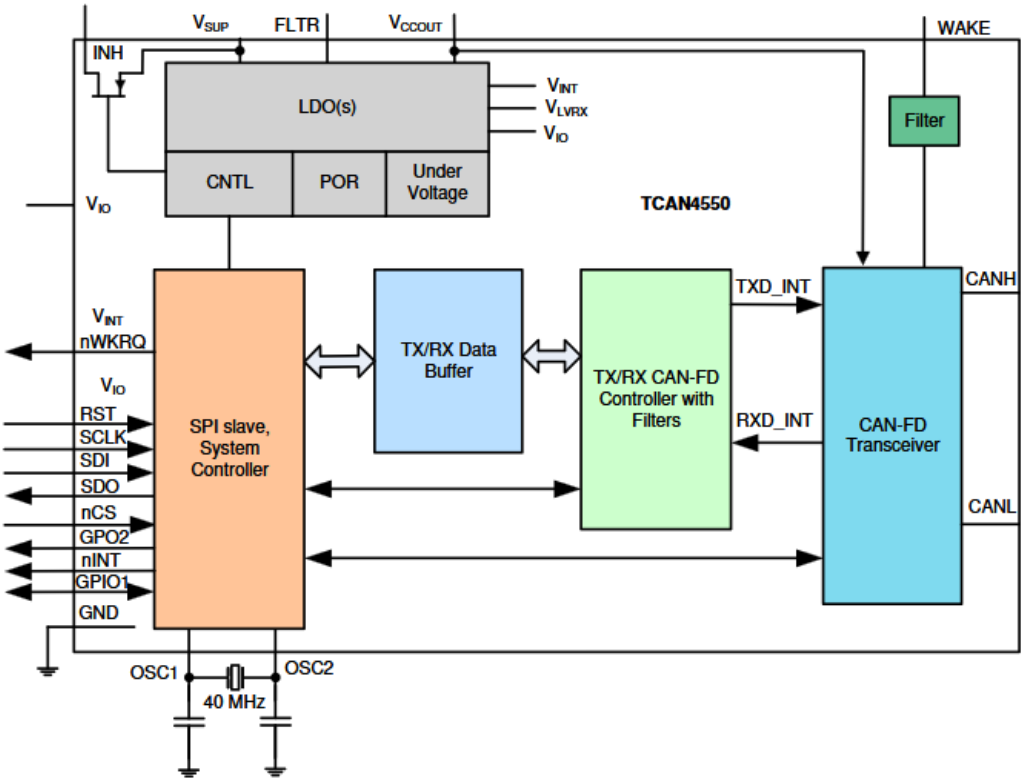


Рисунок 3 – Блок диаграмма TCAN4550

1.3 Выводы

В рамках первой части была изучена архитектура обмена данными на предприятии, выявлены ограничения передачи команд по CAN-шине в микроконтроллер, а также определены риски, связанные с отсутствием аппаратной фильтрации и маскирования входящих сообщений. На основе полученных данных была сформулирована задача, решение которой требует доработки существующей платы, чтобы обеспечить совместимость протоколов и повышение устойчивости системы к внешним воздействиям.

Анализ решений показал, что микросхемы MCP2515 и TCAN4550, несмотря на широкое распространение, не удовлетворяют требованиям, предъявляемым к разрабатываемому преобразователю интерфейсов. Контроллер MCP2515 имеет ограниченную архитектуру: всего два приёмных буфера, минимальное количество фильтров и отсутствие аппаратных механизмов защиты обмена по SPI. Это снижает устойчивость системы при высокой интенсивности обмена. Кроме того, отсутствие ECC-коррекции и CRC-контроля SPI делает MCP2515 менее надёжным в условиях промышленной эксплуатации, где возможны помехи и ошибки памяти.

Микросхема TCAN4550 представляет собой более сложное решение, совмещающее CAN FD-контроллер и встроенный трансивер, который в данной архитектуре платы уже присутствует, поэтому это решение тоже не подходит.

Таким образом, выбор микросхемы MCP2517FD является технически обоснованным и оптимальным для реализации преобразователя логических сигналов между SPI и CAN, обеспечивающего требуемый уровень функциональности.

2 Структура аппаратно-программного комплекса

2.1 Общие сведения

Общая структура аппаратуры представлена на схеме 1 приложения Б. В спецмашине находится устройство, использующее плату, которая дорабатывается в рамках данного курсового проекта.

Шум для маскирования приходит с СБ, который представлен отдельной платой в данном устройстве. СБ представляет из себя плату, реализующую различные криптографические алгоритмы. Шум нужный для маскирования можно запросить, используя протокол взаимодействия. Фрагмент протокола представлен ниже.

КИ, который управляет платой индикации, реализован на контроллере STM32L471RET6, сопрягается с СБ, выполненным на процессоре TMS320VC5509, по стандартному последовательному интерфейсу на скорости 115200 бит/с.

Сопряжение КИ и СБ осуществляется по цепям, приведённым в таблице 1.

Таблица 1 – Цепи последовательного интерфейса

Наименование сигнала	Обозначение	Вх/вых КИ
1 Принимаемая информация КИ от СБ	RX	Вх.
2 Передаваемая информация КИ в СБ	TX	Вых.
Примечание – уровень логических сигналов: логический 0 – 0 В, логическая 1 – + 3,3 В		

Примеры команд, передаваемые из контроллера КИ в СБ, ответные сообщения и их коды приведены в таблице 2.

Таблица 2 – Примеры команд

Команды из КИ в СБ	Код	Команды из СБ в КИ	Код
Запрос времени/даты	0x76	Время/дата (+6 байт)	0x90
Установка времени/даты (+6 байт)	0x77	Выполнено Не выполнено	0x91 0x92
Опрос состояния АРС ОМСК	0x92	АРС исправна АРС не исправна	0x8C 0x8D
АРС ОМСК ВКЛ АРС ОМСК ВЫКЛ	0x96 0x97	Выполнено Не выполнено	0x8E 0x8F
Запрос шума для CAN	0x1F	ПДСЧ ответ (+144 байта)	0x1F

АРМ является средством ввода и отображения управляющей информации. Ниже представлен протокол, который должен быть реализован на плате.

Цифровой стык управления имеет следующие параметры:

- интерфейс – CAN, версия – 2.0b, расширенное ID – 29 бит (ID=0x417300);
- скорость передачи – 1 Мбит/с;
- уровни сигналов: PHY драйвера с питанием 5В (MAX 3053 или аналогичный);
- код длины данных 4 бита (0-8 байт в посылке);
- сигналы – дифференциальные.

Наименование и назначение цепей стыка управления приведено в таблице 1.

Таблица 3 – Цепи стыка

Назначение цепи	Наименование цепи	Аппаратура номер контакта	АРМ номер контакта
Цепь управления 1	CAN-H	1	1
Цепь управления 2	CAN-L	2	2
Экран проводов 1,2	SGND	7	7

Цепь принимаемых и передаваемых сигналов управления должна быть свита попарно и помещена в экранированную оплетку.

Формат команд протокола CAN приведен на рисунке 4.

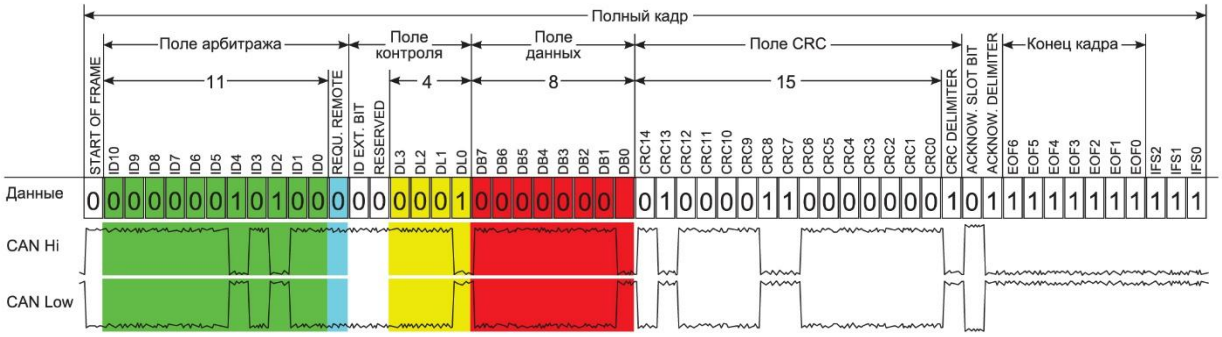


Рисунок 4 – Формат команд CAN

Формат кадра передачи данных в протоколе CAN приведен в таблице

Таблица 4 – Формат кадра CAN

Наименование поля	Длина в битах	Описание
SOF (Start of frame)	1	Начало кадра
Identifier, выделено зеленым	29	Предоставляет приоритет сообщения 29 бит для стандартной CAN
RTR выделено голубым	1	Удаленный запрос передачи
Наименование поля	Длина в битах	Описание
IDE (Identifier extension bit)	1	Бит-идентификатор служит для идентификации расширенного формата
r0	1	Зарезервированный бит для будущего расширения протокола
DLC (Data Length Code), выделено желтым	4	Код длины данных 4 бит для стандартной CAN
Data Field, выделено красным	0–64	Поле данных, передаваемые данные 0–8 байт для стандартного CAN

CRC (Cyclic redundancy check)	15	Контрольная сумма, используется для обнаружения ошибок
CRC Delimiter	1	Бит - разграничитель CRC
ACK (Acknowledgement) slot	1	Область подтверждения. Доминантный бит при сообщении об ошибке; рецессивный бит при отказе от сообщения об ошибке
ACK Delimiter	1	Бит-разграничитель подтверждения
EOF (End of frame)	7	Конец кадра

Пример Команд, передаваемых от АРМ к устройству приведен в таблице 5.

Таблица 5 – Пример команд

АРМ	Аппаратура	Код команды
Запрос состояния 1	Прием и обработка команд	0x20
Запрос состояния 2	Прием и обработка команд	0x21
Передача информации	Прием и обработка команд	0x2B

Для настройки скорости шины CAN в программной части, нужно посчитать какие параметры нужно указать в регистре. Скорость шины настраивается с помощью подсчета сегментов времени передачи одного бита по шине. Сегменты времени бита в шине CAN представлены на рисунке 5.

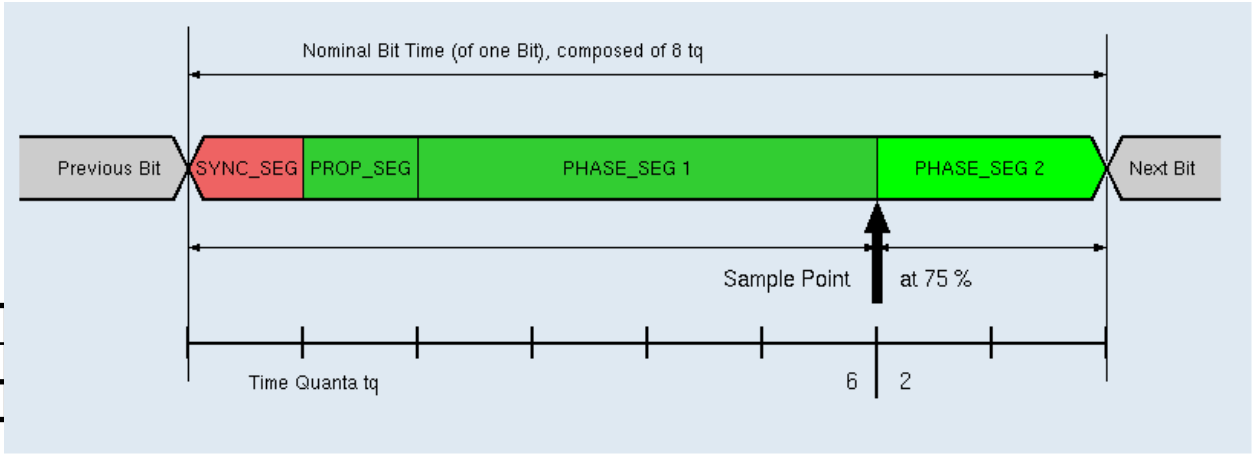


Рисунок 5 – Сегменты времени бита CAN

Сегмент синхронизации (SYNC_SEG) всегда имеет длину 1 квант, его считать не нужно. Сегмент воспроизведения (PROP_SEG) нужен для компенсации задержки на линиях шины. С помощью первого сегмента и второго сегмента настраивается время. Также учитывается момент замера уровня сигнала контроллером, часто ее выбирают в промежутке от 75-87.5% от общего времени передачи бита. При подсчетах значений принято объединять сегмент 1 и сегмент воспроизведения, далее по тексту сегмент 1 будет равен их сумме.

Из вышесказанного следует что нужно посчитать 2 значения, фаза первого сегмента (*x*) и фаза второго сегмента (*y*). Точка замера (*Sample point*) была выбрана равной 85% (*SP*). А так как скорость шины (*f*) считается относительно резонатора, к которому подключена микросхема MCP2517FD, то в формуле буду использовать 20МГц в соответствии с документацией (рисунок 6).

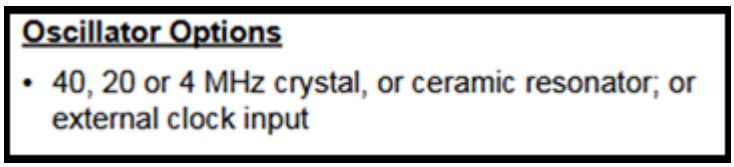


Рисунок 6 – Требования к частоте

Нужная частота в соответствии с протоколом равна 1 Мбит/с (bps). Предделитель (*presc*) не понадобится, он будет равен 1. Чтобы посчитать 2 неизвестных значения, нужно решить систему уравнений.

$$\begin{cases} (x + y + 1) * \dfrac{presc}{f} = \dfrac{1}{bps} \\ \dfrac{x\ +\ 1}{x\ +\ y + 1} = \dfrac{SP}{100} \end{cases}$$

Решив эту систему уравнений, были получены значения x = 16, y = 3.

2.2 Аппаратная часть

Функционал маскирования реализуется с помощью микросхемы XOR. В качестве нее была выбрана sn74aup1T86, образ представлен на рисунке 7.

Микросхема имеет 2 информационных входа над которыми выполняется операция, выход через который получаем результат, а также питание.

На вход A был заведен шум, который выставляется с контроллера. На вход B будут выходить данные с микросхемы mcp2517fd. Маскирование организованно таким образом что за порт B заведена частота SPI и когда выдается символ по интерфейсу, срабатывает прерывание по частоте и выдается бит маски который складывается с битом, который шел по SPI шине.

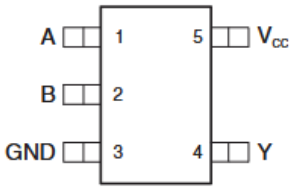


Рисунок 7 – микросхема sn74aup1T86

На вход A был заведен шум, который выставляется с контроллера. На вход B будут выходить данные с микросхемы mcp2517fd. Маскирование организованно таким образом что за порт B заведена частота SPI и когда выдается символ по интерфейсу, срабатывает прерывание по переднему фронту и выдается бит маски который складывается с битом, который шел по SPI шине. Временная диаграмма представлена на рисунке 8.

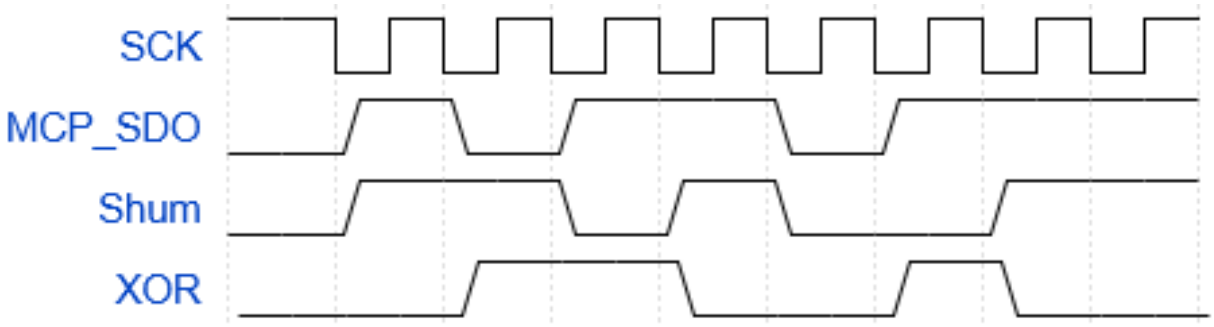


Рисунок 8 – Временная диаграмма

Микросхема MCP2517FD представлена на рисунке 9. TXCAN и RXCAN подключаются к соответствующим выходам преобразователя уровней. CLKO не понадобится. Есть 3 выхода с прерываниями их можно завести на порт C. OSC1 и OSC2 нужны для подключения внешнего тактирования, был поставлен резонатор на 20МГц который поддерживает микросхема как было написано выше. VDD и VSS подключил к положительному напряжению и земле соответственно. EP как написано в документации подключается к земле. SDO, SDI, nCS, SCK это линии SPI подключаются с соответствующими линиями от контроллера. Готовый фрагмент схемы представлен в приложении Б.

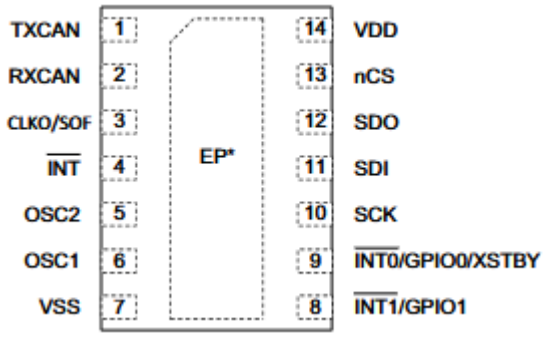


Рисунок 9 – микросхема MCP2517FD

2.3 Программная часть

Перед началом написания программного обеспечения была изучена документация микросхемы MCP2517fd [1,2]. Определился с нужными регистрами, которые нужно настроить. Регистры в данной микросхеме делятся на 2 вида: регистры отвечающие за настройку микросхемы в целом и регистры настройки работы с сообщениями. К первому виду относятся регистры, отвечающие за настройку тактирования микросхемы, настройку выходов, подсчета контрольной суммы памяти. Ко второму виду относятся регистры с помощью, которых настраиваются очереди сообщений, режим работы микросхемы, фильтра и тд. Каждый вид регистров имеет отдельную область памяти. Также в памяти есть отдельная область для хранения сообщений. Карта памяти микросхемы представлена на рисунке 10.

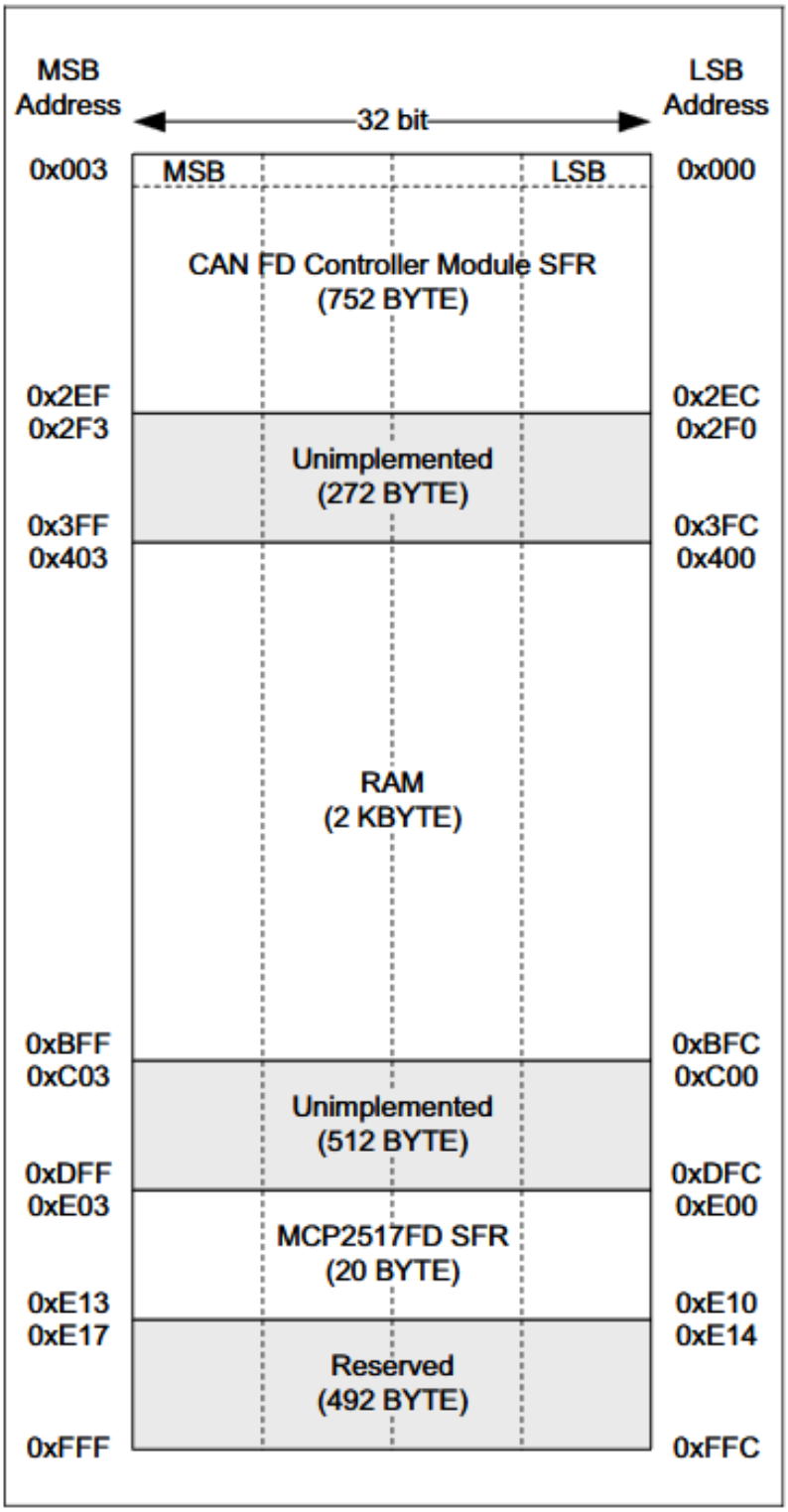


Рисунок 10 – Карта памяти

Зная адреса регистров, можно начать настройку микросхемы. Структура регистра CiCON представлена на рисунке 11. Он используется для настройки режима работы всей микросхемы. Здесь нужны биты с 24 по 26, они отвечают за установление режима работы.

REGISTER 3-6: CiCON – CAN CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0
TXBWS<3:0>				ABAT	REQOP<2:0>		
bit 31							bit 24

R-1	R-0	R-0	R/W-1	R/W-1	R/W-0	R/W-0	R/W-0
OPMOD<2:0>			TXQEN(1)	STEF(1)	SERR2LOM(1)	ESIGM(1)	RTXAT(1)
bit 23							bit 16

U-0	U-0	U-0	R/W-0	R-0	R/W-1	R/W-1	R/W-1
—	—	—	BRSDIS	BUSY	WFT<1:0>		WAKFIL(1)
bit 15							bit 8

U-0	R/W-1	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	PXEDIS(1)	ISOCRCEN(1)	DNCNT<4:0>				
bit 7							bit 0

Рисунок 11 – Регистр CiCON

Нужно выставить режим работы CAN 2.0, так как этот протокол используется в работе. Описание отвечающего за это бита представлено на рисунке12.

bit 26-24 **REQOP<2:0>**: Request Operation Mode bits
000 = Set Normal CAN FD mode; supports mixing of CAN FD and Classic CAN 2.0 frames
001 = Set Sleep mode
010 = Set Internal Loopback mode
011 = Set Listen Only mode
100 = Set Configuration mode
101 = Set External Loopback mode
110 = Set Normal CAN 2.0 mode; possible error frames on CAN FD frames
111 = Set Restricted Operation mode

Рисунок 12 – Бит режима работы

Регистр CiNBTCFG используется для настройки времени передачи бита по интерфейсу CAN. Структура регистра представлена на рисунке 13. Нужно помнить, что к записанным значениям прибавится еще единица. Исходя из этого предделитель равен 0, время первого сегмента 15, время второго сегмента 2. SJW это параметр который показывает максимальное число квантов, на которое может увеличиться время одного бита для синхронизации с другими устройствами. В данном случае можно его поставить в единицу.

REGISTER 3-7: CiNBTCFG – NOMINAL BIT TIME CONFIGURATION REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BRP<7:0>							
bit 31							bit 24

R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-0
TSEG1<7:0>							
bit 23							bit 16

U-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1
—	TSEG2<6:0>						
bit 15							bit 8

U-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1
—	SJW<6:0>						
bit 7							bit 0

Рисунок 13 – Регистр CiNBTCFG

Регистр CiINT используется для настройки прерываний. Структура регистра представлена на рисунке 14. Здесь нужно установить бит 17 отвечающий за прерывание при получении нового сообщения.

REGISTER 3-13: CiINT – INTERRUPT REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IVMIE	WAKIE	CERRIE	SERRIE	RXOVIE	TXATIE	SPICRCIE	ECCIE
bit 31							bit 24

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	TEFIE	MODIE	TBCIE	RXIE	TXIE
bit 23							bit 16

HS/C-0	HS/C-0	HS/C-0	HS/C-0	R-0	R-0	R-0	R-0
IVMIF(1)	WAKIF(1)	CERRIF(1)	SERRIF(1)	RXOVIF	TXATIF	SPICRCIF	ECCIF
bit 15							bit 8

U-0	U-0	U-0	R-0	HS/C-0	HS/C-0	R-0	R-0
—	—	—	TEFIF	MODIF(1)	TBCIF(1)	RXIF	TXIF
bit 7							bit 0

Рисунок 14 – Регистр CiINT

Регистр CiFIFOCONm для настройки очередей сообщений как на отправку, так и на получение. Структура регистра представлена на рисунке 15. Для настройки очередей нужны биты 29-31 для того, чтобы ограничить размер полезной нагрузки до 8 байт, биты 24-28, отвечающие за размер очереди, биты 21-22, чтобы команда посылалась бесконечно, пока не будет получено подтверждение, а также бит 7 для выбора режима очереди.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PLSIZE<2:0>(1)			FSIZE<4:0>(1)				
bit 31							bit 24

U-0	R/W-1	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TXAT<1:0>		TXPRI<4:0>				
bit 23							bit 16

U-0	U-0	U-0	U-0	U-0	S/HC-1	R/W/HC-0	S/HC-0
—	—	—	—	—	FRESET(3)	TXREQ(2)	UINC
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TXEN(1)	RTREN	RXTSEN(1)	TXATIE	RXOVIE	TFERFFIE	TFHRFHIE	TFNRFNIE
bit 7							bit 0

Рисунок 15 – Регистр CiFIFOCONm

Регистр CiFLTCONm и Регистр CiFLTOBJm для настройки фильтра принятых сообщений. Структура регистров представлена на рисунках 16 и 17.

В регистре C1FLTCON1 нужны биты 7 для включения фильтра и 0-4 для привязки фильтра к очереди, то есть куда будут помещаться сообщения если пройдут фильтр.

REGISTER 3-31: CiFLTCONm – FILTER CONTROL REGISTER m, (m = 0 TO 7)

R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTEN3	—	—	F3BP<4:0>(1)				
bit 31							bit 24

R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTEN2	—	—	F2BP<4:0>(1)				
bit 23							bit 16

R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTEN1	—	—	F1BP<4:0>(1)				
bit 15							bit 8

R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTEN0	—	—	F0BP<4:0>(1)				
bit 7							bit 0

Рисунок 16 – Регистр CiFLTCONm

В регистре CiFLTOBJm нужно выбрать расширенный id в соответствии с протоколом, а также сам id, сообщения с которым нужно пропускать дальше, а остальные отбрасывать.

REGISTER 3-32: CiFLTOBJm – FILTER OBJECT REGISTER m,(m = 0 TO 31)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	EXIDE	SID11	EID<17:13>				
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EID<12:5>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EID<4:0>					SID<10:8>		
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SID<7:0>							
bit 7							bit 0

Рисунок 17 – Регистр CiFLTOBJm

Настройка микросхемы начинается с настройки самого интерфейса spi для взаимодействия ней. Для настройки и работы с spi использовалась библиотека HAL [5,6], она обеспечивает должное удобство, быстроту разработки и нужную производительность.

На рисунке 18 представлен фрагмент документации, где показано, что MCP2517 поддерживает режимы 0-0 и 1-1.

SPI Interface

- Up to 20 MHz SPI clock speed
- Supports SPI modes 0,0 and 1,1

Рисунок 18 – Ограничения по частоте SPI

Для управления микросхемой нужно установить режим мастера на модуль spi микроконтроллера, а также полнодуплексный режим, так как микросхема его поддерживает [3,4]. В соответствии с документацией настроен режим 3 SPI то есть CPOL=1, CPHA=1. Настроенная структура SPI выглядит так:

```
hspi3.Instance = SPI3;
hspi3.Init.Mode = SPI_MODE_MASTER;
hspi3.Init.Direction = SPI_DIRECTION_2LINES;
hspi3.Init.DataSize = SPI_DATASIZE_8BIT;
hspi3.Init.CLKPolarity = SPI_POLARITY_HIGH;
hspi3.Init.CLKPhase = SPI_PHASE_2EDGE;
```

```
  hspi3.Init.NSS = SPI_NSS_SOFT;
  hspi3.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_128;
  hspi3.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi3.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi3.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi3.Init.CRCPolynomial = 7;
  hspi3.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi3.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
```

Рассмотрим общий формат команд, он представлен на рисунках 19.

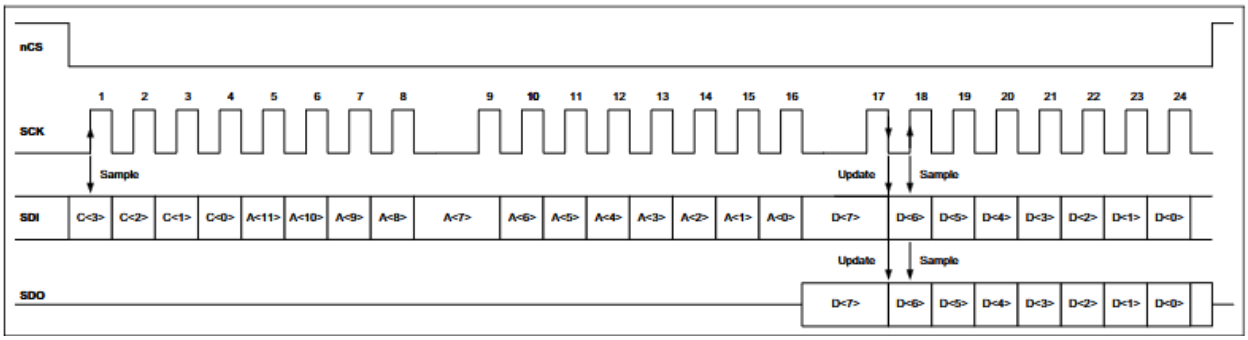


Рисунок 19 – Формат команд MCP2517FD

Формат команд чтения и записи представлен на рисунке 20.

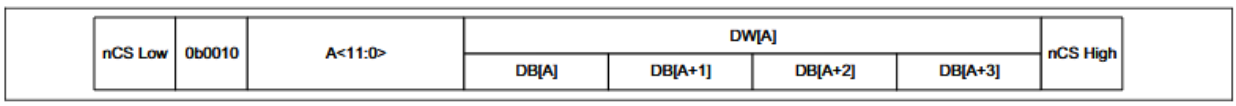


Рисунок 20 – Формат команд чтения и записи

Если посылается команда записи или чтения, то сначала передаются 4 бита номера операции, затем 12 бит адреса, а потому уже данные. Если используется команда перезагрузки, то нужно отправить 2 нулевых байта. Ниже приведена функция записи значения в регистр.

```
void MCP2517_WriteRegister(uint16_t addr, uint32_t value)
{
  Can_CS_Low(); //CS low

  MCP2517_Send_Byte((MCP_INST_WRITE_DATA << 4) | ((addr >> 8) &
0xF));
  MCP2517_Send_Byte(addr & 0xFF);
```

```
  MCP2517_Send_Byte(value & 0xFF);
  MCP2517_Send_Byte((value >> 8) & 0xFF);
  MCP2517_Send_Byte((value >> 16) & 0xFF);
  MCP2517_Send_Byte((value >> 24) & 0xFF);
  Can_CS_High(); // CS high
}
```

Для начала конфигурации нужно перевести микросхему в режим конфигурации, для этого нужно в регистр CiCON записать соответствующее значение, а затем поставить режим CAN 2.0 для начала работы с сообщениями.

Значения для каждого регистра хранятся в соответствующем массиве, а затем записываются в микросхему одной функцией. Ниже приведен пример массива конфигурации времени бита CAN.

```
 uint32_t C1NBTCFG_data[4] = {
0x0 << 24, // BRP (Prescaler)
15 << 16, //TSEG1(Prop + Phase1)
2  << 8,   //TSEG2(Phase2)
0x1         //SJW
};
```

Для конфигурирования микросхемы нужно поднять CS интерфейса SPI, перевести микросхему в режим конфигурации и перезагрузить, затем записать нужные регистры и выставить режим CAN 2.0. Функция конфигурации микросхемы выглядит так:

```
void MCP2517_Init(void)
{
  MCP2517_EnterConfigMode();
  MCP2517_Reset();
  MCP2517_ConfigurationFilters();
  MCP2517_EnterNormalMode();
}
```

После конфигурации микросхемы можно приступить к написанию функций передачи и приема сообщений. Организации памяти сообщений представлена на рисунке 21. В данной работе достаточно использовать 2 сконфигурированные очереди FIFO 1 и FIFO 2, остальные не понадобятся

Чтобы передать или прочитать сообщения нужно обратиться по соответствующим адресам этих очередей.

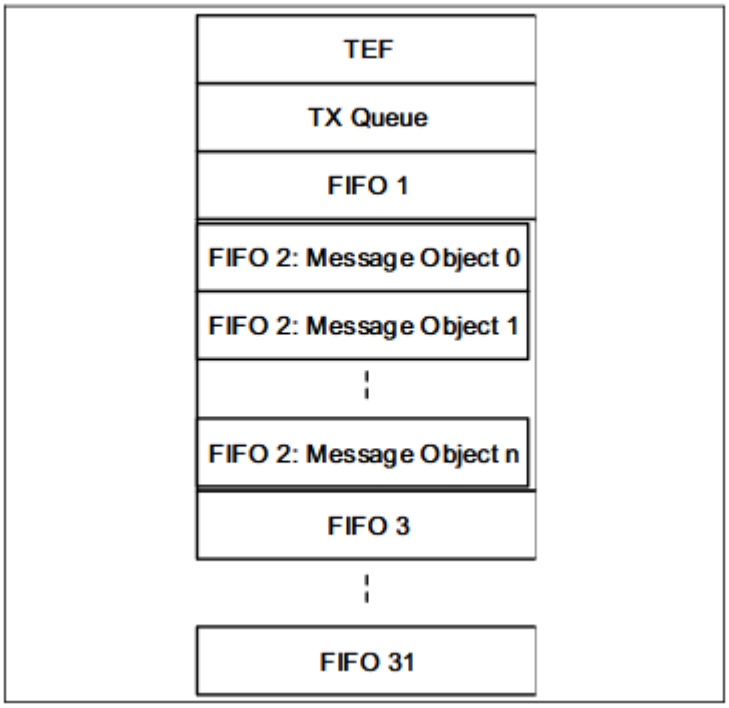


Рисунок 21 – Порядок очередей в памяти

Чтобы обратиться к следующем сообщению нужно к адресу начала памяти сообщений прибавить смещение. За смещение отвечает регистр очереди CiFIFOUAm, его описание представлено на рисунке 22.

bit 31-0 **FIFOUA<31:0>:** FIFO User Address bits
TXEN = 1 (FIFO is configured as a Transmit FIFO)
A read of this register will return the address where the next message is to be written (FIFO head).
TXEN = 0 (FIFO is configured as a Receive FIFO)
A read of this register will return the address where the next message is to be read (FIFO tail).

Рисунок 22 – Бит адреса следующего сообщения

Чтобы прочитать следующее сообщение в регистре очереди есть бит, показывающий заполнена ли очередь, если она сконфигурирована на передачу сообщений. Если же очередь, сконфигурированная на прием сообщений, то бит показывает есть ли новое сообщение. Этот бит будет использоваться для опроса состояния очереди. Описание этого бита представлено на рисунке 23.

bit 0 **TFNRFNIF:** Transmit/Receive FIFO Not Full/Not Empty Interrupt Flag bit
TXEN = 1 (FIFO is configured as a Transmit FIFO)
Transmit FIFO Not Full Interrupt Flag
1 = FIFO is not full
0 = FIFO is full
TXEN = 0 (FIFO is configured as a Receive FIFO)
Receive FIFO Not Empty Interrupt Flag
1 = FIFO is not empty, contains at least one message
0 = FIFO is empty

Рисунок 23 – Описание бита состояния очереди

Далее были изучены форматы сообщений, которые понимает эта микросхема. Формат сообщений представлен на рисунках 24 и 25. Важно побитно соблюсти формат сообщений, чтобы все работало. В сообщениях, которые отправляются, сначала пишется id, затем идет последовательность бит, которая не нужна в данной работе, затем флаги, потом 4 бита размер данных, а в конце сами данные.

Формат сообщений, которые принимаются, немного отличаются. Сначала идет id сообщения, затем 2 байта не значащие биты. Следом идет 5 битов номера фильтра который пропустил сообщение, 2 не значащих бита и флаги, после чего размер данных. В самом конце идут полученные данные.

TABLE 3-5: TRANSMIT MESSAGE OBJECT (TXQ AND TX FIFO)

Word		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
T0	31:24	—	—	SID11	EID<17:6>				
	23:16	EID<12:5>							
	15:8	EID<4:0>					SID<10:8>		
	7:0	SID<7:0>							
T1	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	SEQ<6:0>							ESI
	7:0	FDF	BRS	RTR	IDE	DLC<3:0>			
T2 (1)	31:24	Transmit Data Byte 3							
	23:16	Transmit Data Byte 2							
	15:8	Transmit Data Byte 1							
	7:0	Transmit Data Byte 0							
T3	31:24	Transmit Data Byte 7							
	23:16	Transmit Data Byte 6							
	15:8	Transmit Data Byte 5							
	7:0	Transmit Data Byte 4							
Ti	31:24	Transmit Data Byte n							
	23:16	Transmit Data Byte n-1							
	15:8	Transmit Data Byte n-2							
	7:0	Transmit Data Byte n-3							

Рисунок 24 – Формат отправляемых сообщений

TABLE 3-6: RECEIVE MESSAGE OBJECT

Word		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
R0	31:24	—	—	SID11	EID<17:6>				
	23:16	EID<12:5>							
	15:8	EID<4:0>					SID<10:8>		
	7:0	SID<7:0>							
R1	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	FILHIT<4:0>					—	—	ESI
	7:0	FDF	BRS	RTR	IDE	DLC<3:0>			
R2 (2)	31:24	RXMSGTS<31:24>							
	23:16	RXMSGTS<23:16>							
	15:8	RXMSGTS<15:8>							
	7:0	RXMSGTS<7:0>							
R3 (1)	31:24	Receive Data Byte 3							
	23:16	Receive Data Byte 2							
	15:8	Receive Data Byte 1							
	7:0	Receive Data Byte 0							
R4	31:24	Receive Data Byte 7							
	23:16	Receive Data Byte 6							
	15:8	Receive Data Byte 5							
	7:0	Receive Data Byte 4							
Ri	31:24	Receive Data Byte n							
	23:16	Receive Data Byte n-1							
	15:8	Receive Data Byte n-2							
	7:0	Receive Data Byte n-3							

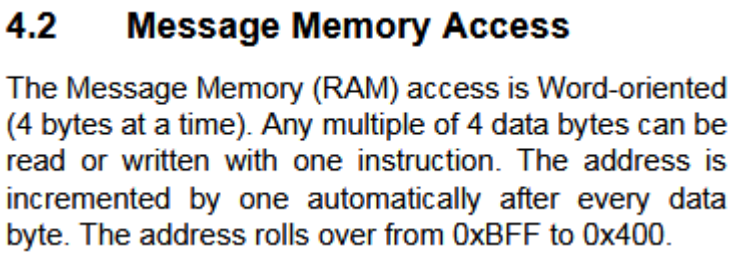
Рисунок 25 – Формат принимаемых сообщений

Для организации сообщений на контроллере используются кольцевые буферы, один буфер для хранения сообщений на отправку, другой для хранения принятых сообщений. Ниже представлены функции взаимодействия с ними.

```
void CBufPut(struct CBuf *Buf, uint8_t  data)
{
  (*Buf).Ptr[ (*Buf).PtrW ]=data;
  (*Buf).PtrW++;
  (*Buf).Cnt++;
  if( (*Buf).PtrW==(*Buf).Len )
  (*Buf).PtrW=0;
}
uint8_t CBufGet(struct CBuf *Buf)
{
  uint8_t data;
  data=(*Buf).Ptr[ (*Buf).PtrR ];
  (*Buf).PtrR++;
  (*Buf).Cnt--;
  if( (*Buf).PtrR==(*Buf).Len )
```

```
  (*Buf).PtrR=0;
  return data;
}
```

Из документации, фрагмент которой представлен на рисунке 26, известно, что каждый записанный пакет должен быть кратен 4 [3]. Соответственно нужно корректировать размер сообщений.



4.2 Message Memory Access

The Message Memory (RAM) access is Word-oriented (4 bytes at a time). Any multiple of 4 data bytes can be read or written with one instruction. The address is incremented by one automatically after every data byte. The address rolls over from 0xBFF to 0x400.

Рисунок 26 – Требования к формату сообщений

Когда сообщение записано в очередь на отправку, нужно увеличить счетчик сообщений в регистре очереди, а если сообщение считано уменьшить его. Также при использовании очереди на отправку нужно выставить бит запроса на передачу, чтобы микросхема отправила сообщения.

При отправке сообщения формируется правильная последовательность частей пакета на передачу, записываю получившийся массив в очередь микросхемы и выставляю нужные регистры очереди. Чтение сообщения организованно чуть сложнее. Сначала смотрю в регистре есть ли новое сообщение, если есть читаю сообщение из очереди. Затем раскладываю на составные части. Чтобы размаскированная команда не попала в память вызывается функция, которая сравнивает все возможные замаскированные команды с той, что пришла и если такая нашлась, то размаскирую и кладу в буфер, если нет, то игнорирую. Фрагмент кода приведен ниже.

```
    if (TruthCommands(&tmp[0]))
    {
      for (k=0; k<8; k++) tmp[k+8] = tmp[k+8] ^ TRUTH[k+8];
//unmask tmp

      test_cnt = (test_cnt ^ TRUTH[4]);   //unmask cnt
      test_cnt &= 0xF;
```

```
      for (k=0; k<8; k++) REC_FRAME[k]=0xFF;
      for (k=0; k<test_cnt; k++) REC_FRAME[k] = tmp[k+8];

      ICANBufPut(REC_FRAME);
    }
```

Также стоит учесть, что значения, прочитанные из регистров, нужно сразу записать в память, для этого выставляется флаг при чтении из памяти сообщений микросхемы. Соответственно если флаг не выставлен сообщение сразу размаскируется.

```
  if (Truth_flag) {
    TRUTH[(Truth_flag-1)] = shum_byte_r;
    Truth_flag++;
  }
    else byte = (byte ^ shum_byte_r);
```

Чтобы узнать пришло ли новое сообщение, используется прерывание по линии 6 порта C. Для работы с прерываниями в контроллерах STM32 используется контроллер внешних прерываний EXTI [7]. При возникновении прерывания выставляется флаг, который означает то нужно вызвать функцию чтения следующего сообщения.

```
void EXTI9_5_IRQHandler(void)
{
  if (EXTI->PR1 & EXTI_PR1_PIF6)
  {
    CAN_Receive_Flag=1;
    EXTI->PR1 |= EXTI_PR1_PIF6;
  }
}
```

Также нужно реализовать маскирования каждого сообщения как было описано выше, шум для маскирования приходит от спецблока по UART. Для хранения шума используется 2 массива на 432 байта. После подачи питания на прибор, в предпусковом контроле заполняться оба массива, когда первый кончится, маска будет браться из второго, а первый в это время заполняться

новыми значениями. Фрагмент функции предпускового контроля с ожиданием шума при веден ниже.

```
  for (;;)
  {
    if ((CAN_StateNoise_Flag_1 != 3)&&(!CAN_WaitAnswer_Flag))
InquiryCanNoise();
    if (OmspBuf.Cnt) SendComm();
    if (ImspBuf.PtrW!=ImspBuf.PtrR) TestComm();

    if ((CAN_StateNoise_Flag_1 ==
3)&&(ImspBuf.PtrW==ImspBuf.PtrR)&&(OmspBuf.Cnt==0)) break;
    if (GameOver) break;
  }
  CAN_TypePartNoise_Flag=0;
  CAN_StateNoise_Flag_2=0;
```

Шум для маскирования выставляется побитно каждое прерывание, которое срабатывает по частоте. Биты для размаскирования читаются также с порта, чтобы если на линии случались ошибки от них, не страдала передача команд, то есть каждый пакет размаскировался всегда ровно тем, чем и замаскировался. Фрагмент кода обработчика прерывания приведен ниже.

```
if ((GPIOA->IDR & GPIO_PIN_2)) shum_byte_r |= shift_bit;
   else shum_byte_r &= (~shift_bit);
   shift_bit = ((shift_bit>>1)&0xff);

if ((shum_byte & 0x80)) GPIOA->BSRR = GPIO_PIN_2;
   else GPIOA->BRR = GPIO_PIN_2;
   shum_byte = ((shum_byte<<1)&0xff);
```

Для получения шума от СБ вызывается функция, которая отправляет команду «Запрос шума для CAN» из протокола.

```
void InquiryCanNoise(void)
{
FormPacketKK(OUT_KK_INQ_NOISE_CAN, TEMP_OUT_PACKET, OUT_PACKET,
0);
CAN_WaitAnswer_Flag = 1;
```

```
}
```

3 Тестирование

При демонстрации работы устройства использовалась отладочная среда IDE IAR Embedded Workbench [8], кабель с преобразователем USB-CAN Titan CAN Test для передачи команд на плату и программу Titan CAN Test для отправки команд.

В качестве проверки была смоделирована передача команд из протокола. Использованные команды представлены в таблице 6.

Таблица 6 – Трестирующиеся команды

№	АРМ	Аппаратура	Код команды
1	Запрос состояния 1	Прием и обработка команд	0x20
2	Передача информации	Прием и обработка команд	0x2B

Команды 1 и 2 не требуют отправки полезной нагрузки. Формат команды 3 представлен на рисунке 27. Содержит 5 цифр в ASCII коде.

1б	2б	3б	4б	5б	6б
0x2B	1 цифра	2 цифра	3 цифра	4 цифра	5 цифра

Рисунок 27– Формат команды 3.

Подключил кабель с преобразователем к разъему шины CAN и ПЭВМ. На ПЭВМ открыл программу Titan CAN Test и настроил в соответствии с рисунком 28. Далее прошил плату и запустил программу в режиме отладки. Точки остановки были поставлены в функции, которая принимает сообщения.

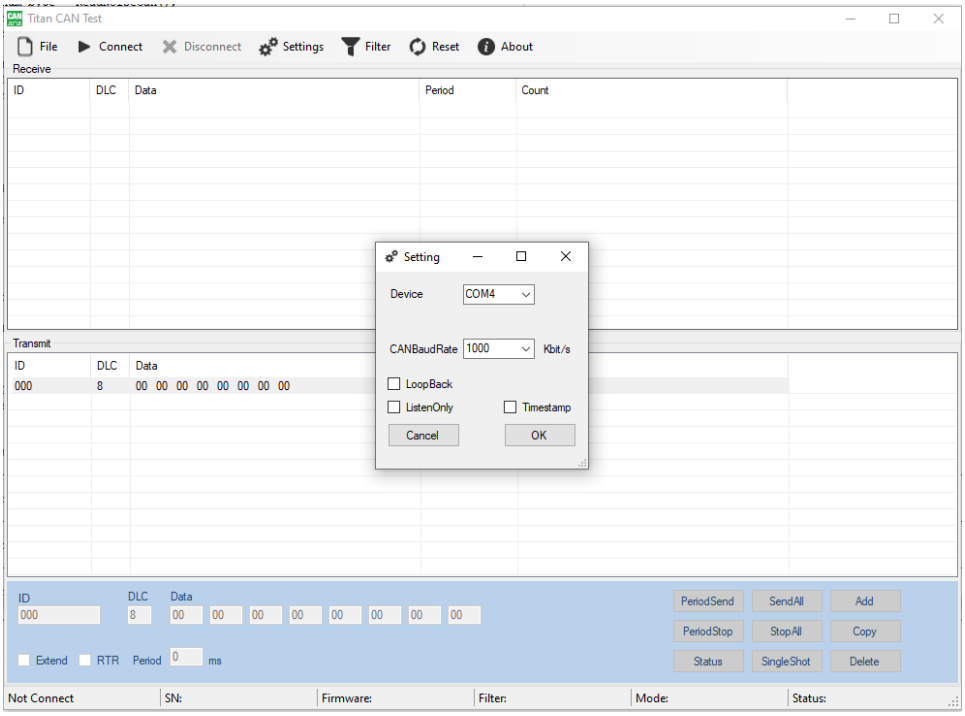


Рисунок 28 – Настройка Titan CAN Test

3.1 Тест 1

В первом тесте отправляется команда с кодом 0x20 (рисунок 29).

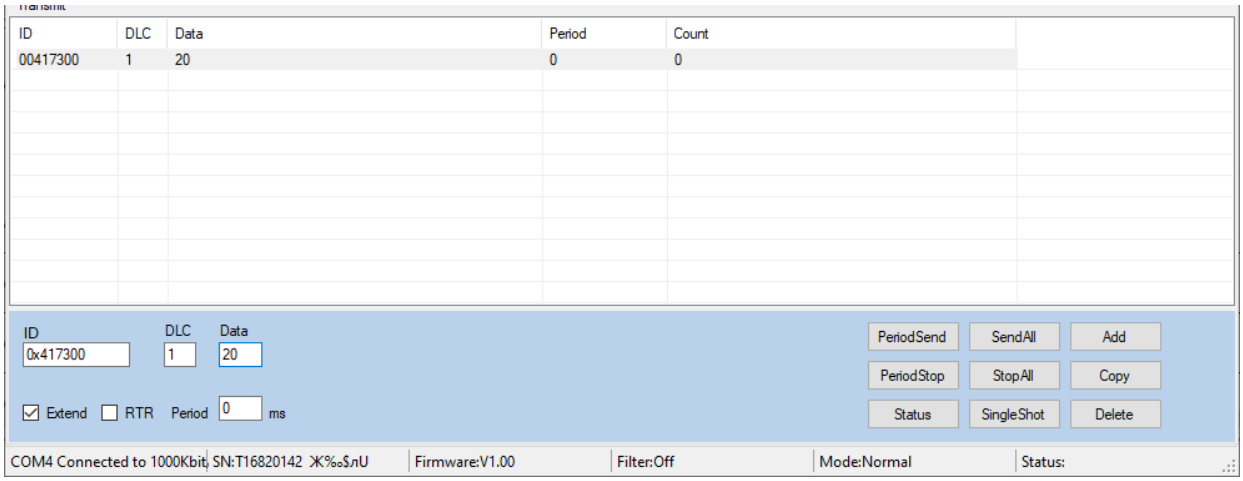


Рисунок 29 – Отправление первой команды.

Первая остановка чтобы посмотреть пакет перед размаскированием (рисунок 30).

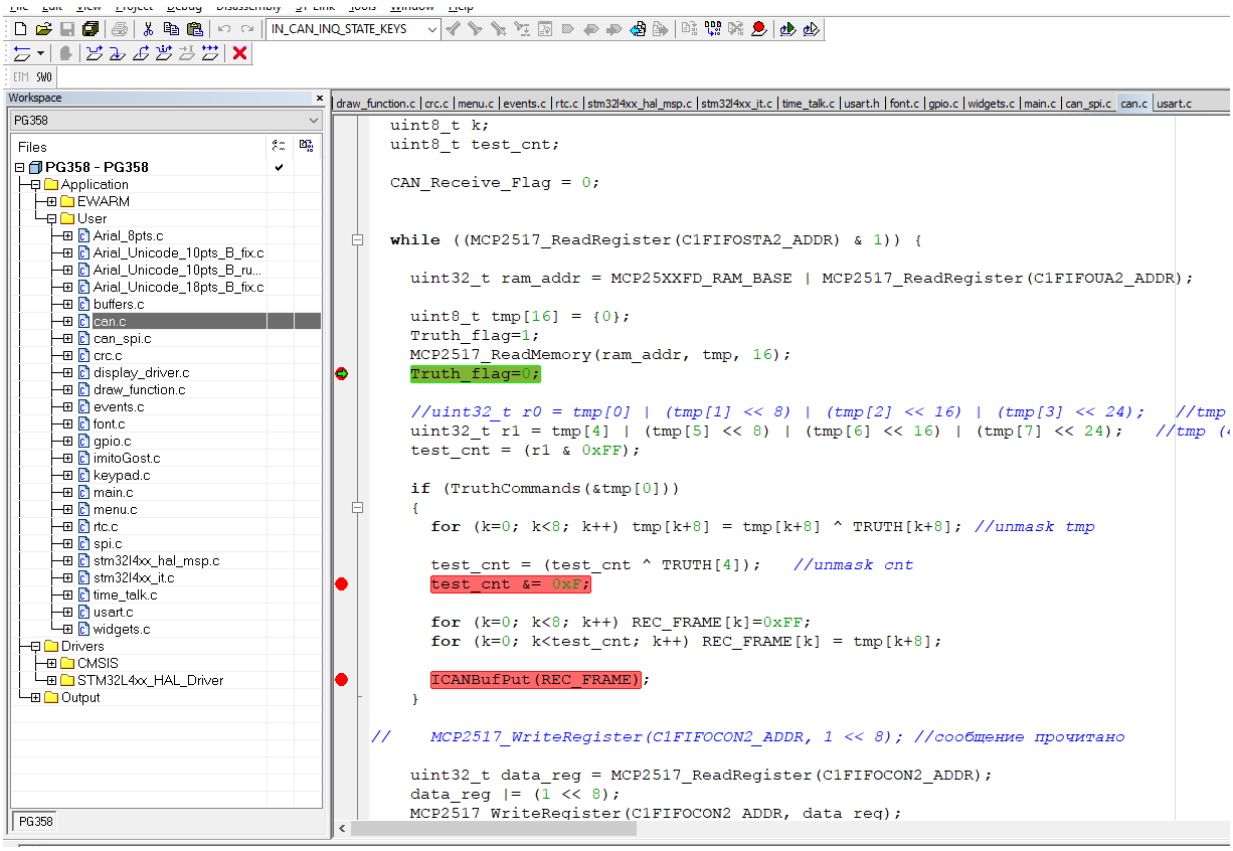


Рисунок 30 – Точка остановки

Принятое сообщение представлено на рисунке 31.

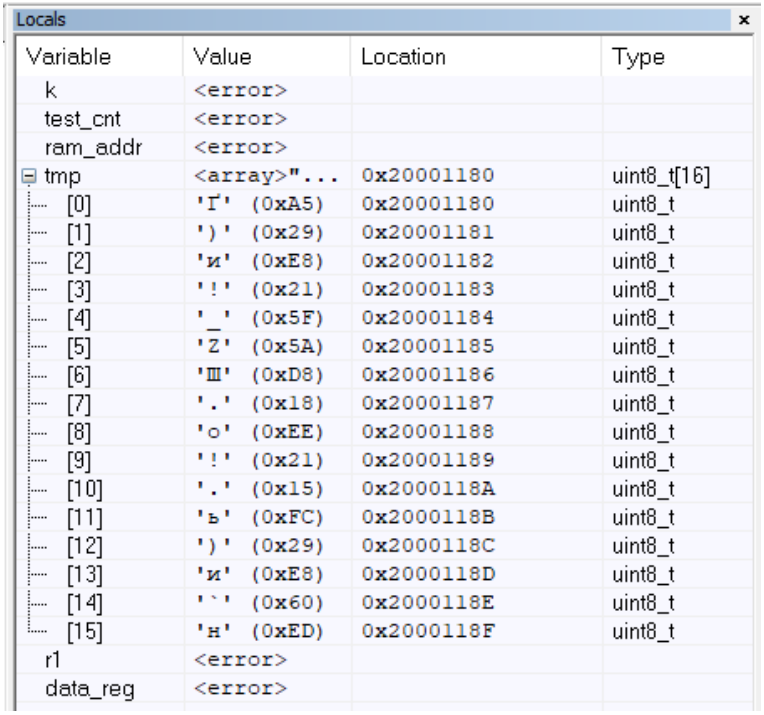


Рисунок 31 – Замаскированный пакет

Далее размаскируется команда так как пришла команда и показана полезная нагрузка на рисунке 32.

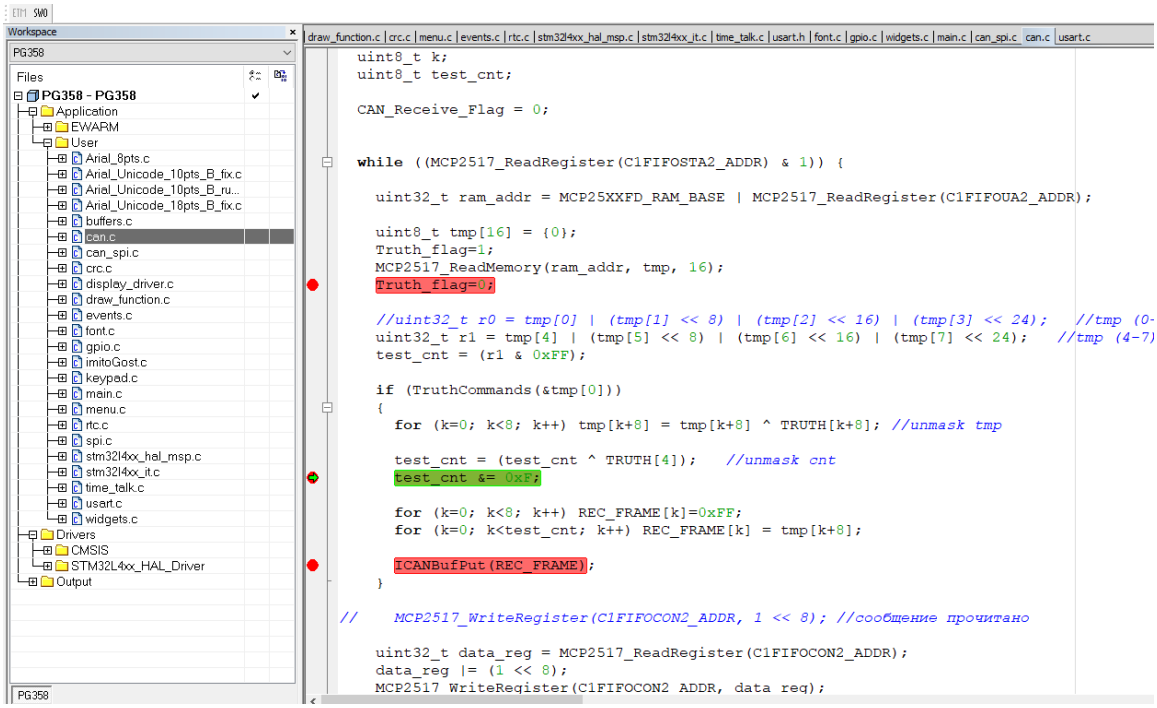
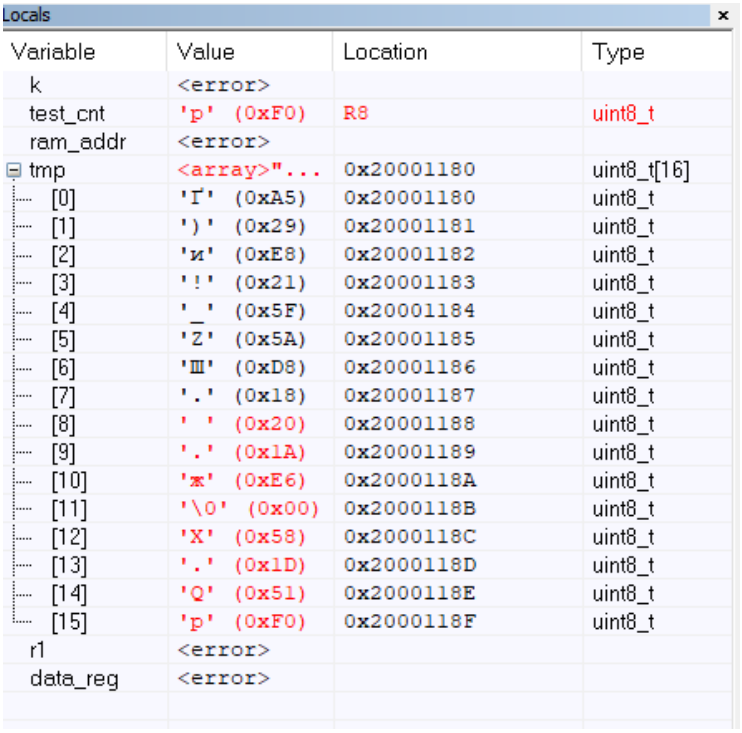


Рисунок 32 – Размаскирование данных

Размаскированные данные представленны на рисунке 33.



Variable	Value	Location	Type
k	<error>		
test_cnt	'p' (0xF0)	R8	uint8_t
ram_addr	<error>		
tmp	<array>"...	0x20001180	uint8_t[16]
[0]	'Г' (0xA5)	0x20001180	uint8_t
[1]	')' (0x29)	0x20001181	uint8_t
[2]	'и' (0xE8)	0x20001182	uint8_t
[3]	'!' (0x21)	0x20001183	uint8_t
[4]	'_' (0x5F)	0x20001184	uint8_t
[5]	'Z' (0x5A)	0x20001185	uint8_t
[6]	'Ш' (0xD8)	0x20001186	uint8_t
[7]	'.' (0x18)	0x20001187	uint8_t
[8]	' ' (0x20)	0x20001188	uint8_t
[9]	'.' (0x1A)	0x20001189	uint8_t
[10]	'ж' (0xE6)	0x2000118A	uint8_t
[11]	'\0' (0x00)	0x2000118B	uint8_t
[12]	'X' (0x58)	0x2000118C	uint8_t
[13]	'.' (0x1D)	0x2000118D	uint8_t
[14]	'Q' (0x51)	0x2000118E	uint8_t
[15]	'p' (0xF0)	0x2000118F	uint8_t
r1	<error>		
data_reg	<error>		

Рисунок 34 – Размаскированные данные

На следующей контрольной точке полученная команда записывается в массив для дальнейшей обработки.

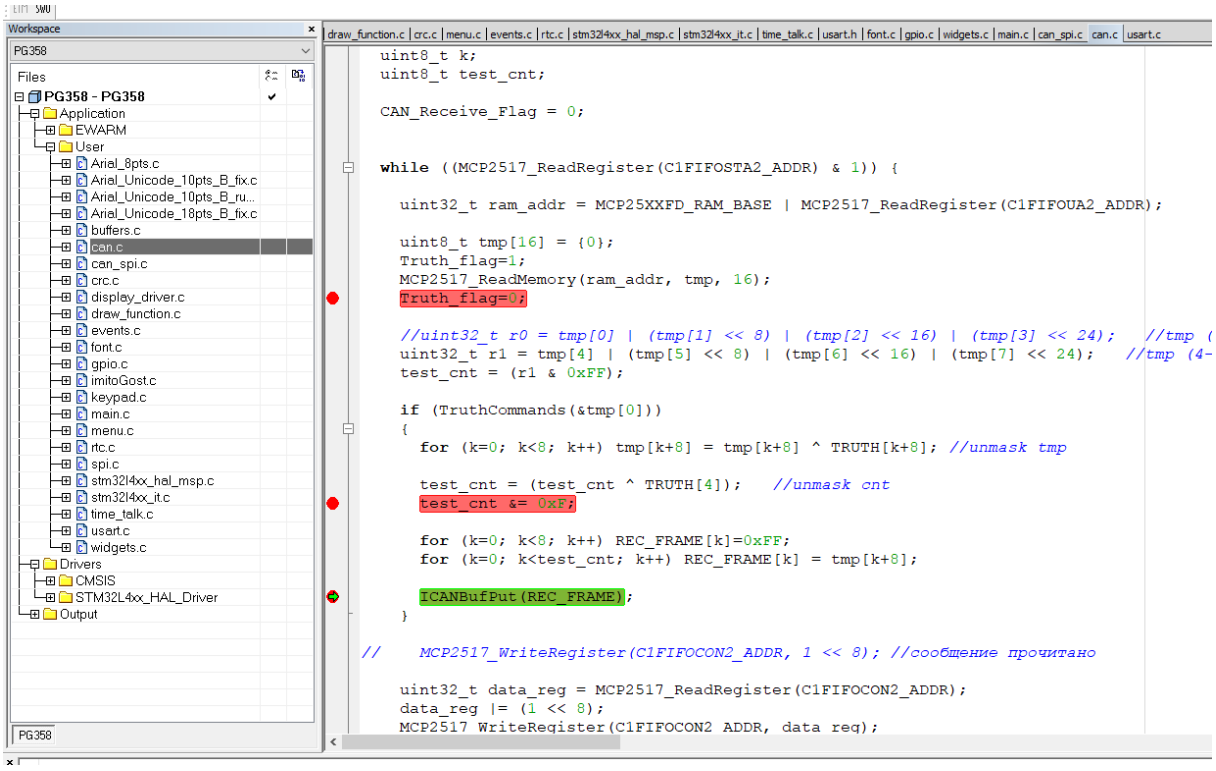


Рисунок 34 – Третья точка остановки

Принятая команда соответствует той, что была отправлена (рисунок 35).

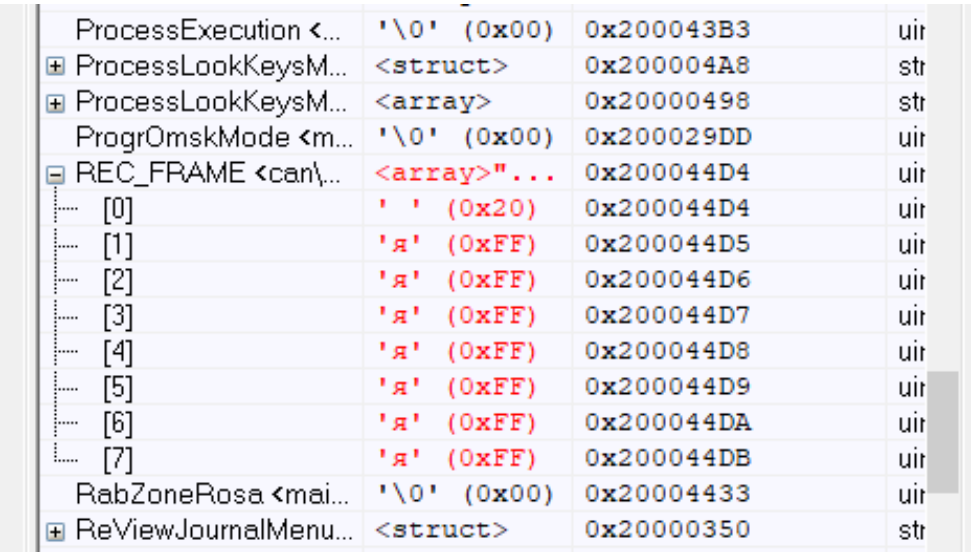


Рисунок 35 – Принятая команда

Так же был получен ответ на эмулятор, после обработки принятой команды. Команда представлена на рисунке 36.

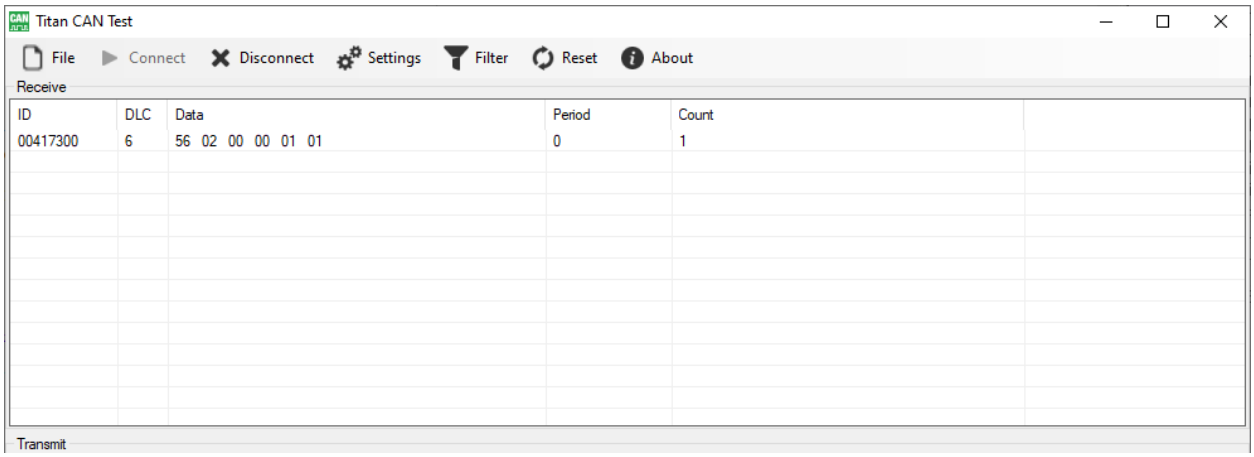


Рисунок 36 – Отправленный ответ с контроллера

3.2 Тест 2

Теперь отправляется вторая команда, которая подразумевает отправку 5 цифр из представления ASCII, она показана на рисунке 37.

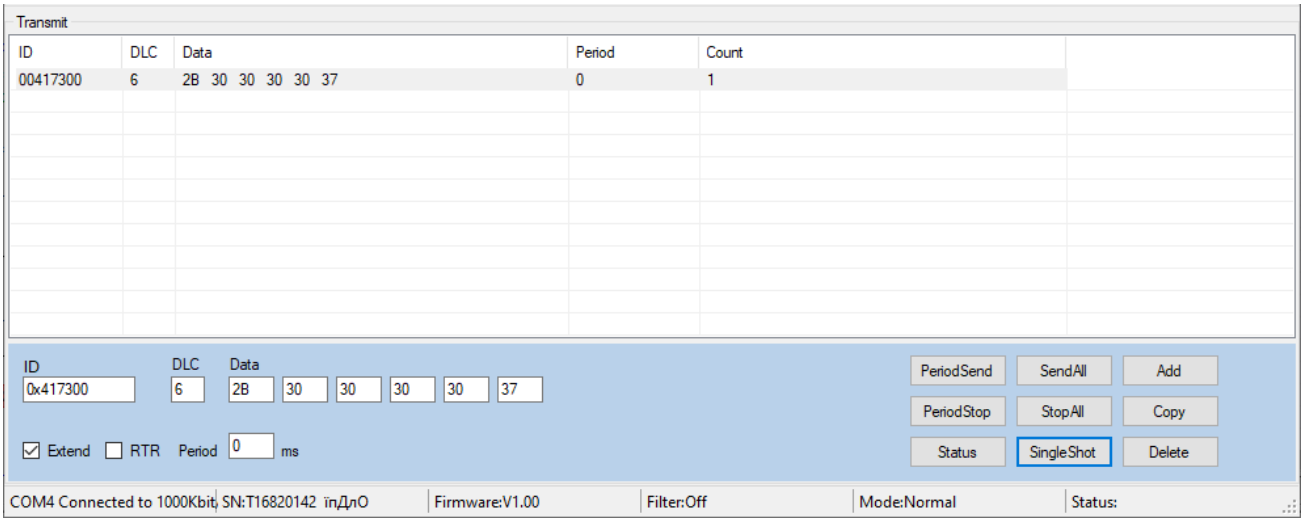


Рисунок 37 – Вторая команда

Принятый пакет перед размаскированием представлен на рисунке 38.

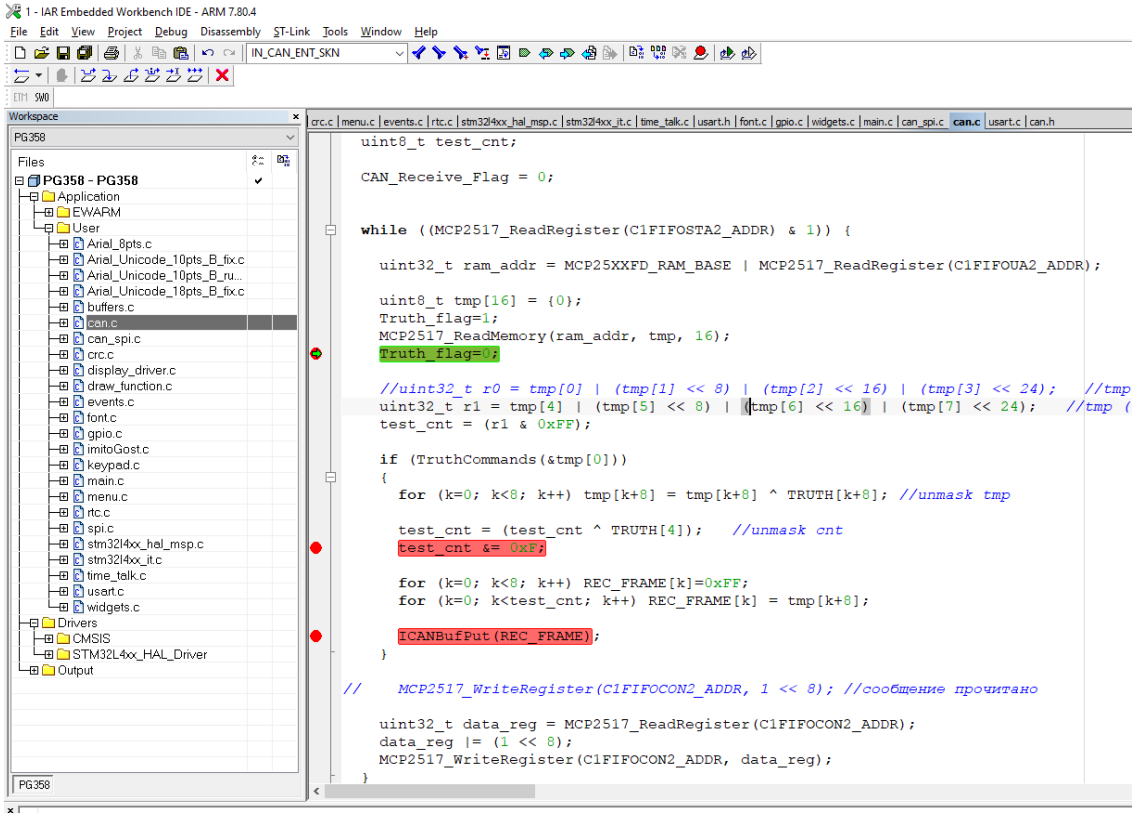
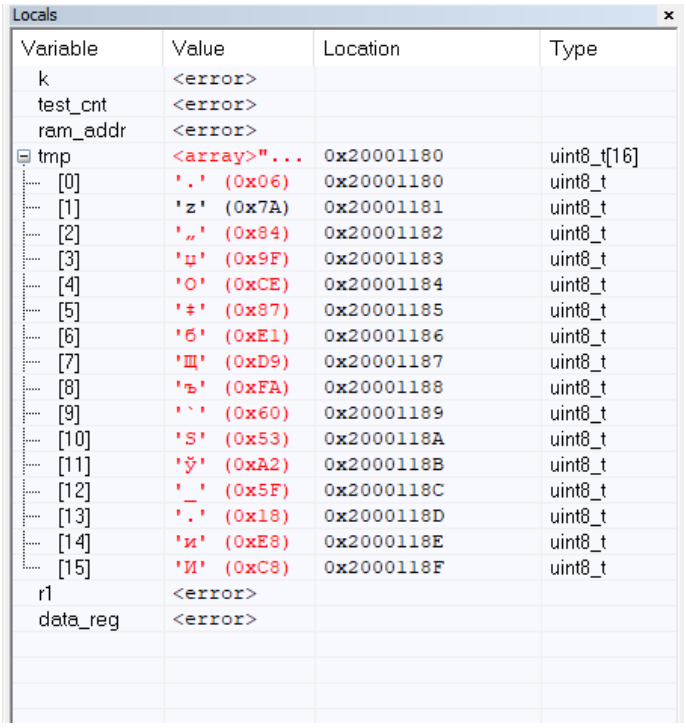


Рисунок 38 – Первая точка остановки

Принятое сообщение представлено на рисунке 39.



Locals

Variable	Value	Location	Type
k	<error>		
test_cnt	<error>		
ram_addr	<error>		
tmp	<array>"...	0x20001180	uint8_t[16]
[0]	'.' (0x06)	0x20001180	uint8_t
[1]	'z' (0x7A)	0x20001181	uint8_t
[2]	'„' (0x84)	0x20001182	uint8_t
[3]	'џ' (0x9F)	0x20001183	uint8_t
[4]	'О' (0xCE)	0x20001184	uint8_t
[5]	'‡' (0x87)	0x20001185	uint8_t
[6]	'б' (0xE1)	0x20001186	uint8_t
[7]	'Щ' (0xD9)	0x20001187	uint8_t
[8]	'ъ' (0xFA)	0x20001188	uint8_t
[9]	'`' (0x60)	0x20001189	uint8_t
[10]	'S' (0x53)	0x2000118A	uint8_t
[11]	'ў' (0xA2)	0x2000118B	uint8_t
[12]	'_' (0x5F)	0x2000118C	uint8_t
[13]	'.' (0x18)	0x2000118D	uint8_t
[14]	'и' (0xE8)	0x2000118E	uint8_t
[15]	'И' (0xC8)	0x2000118F	uint8_t
r1	<error>		
data_reg	<error>		

Рисунок 39 – Замаскированный пакет

Далее размаскируется команда так как пришла команда и полезная нагрузка показана на рисунке 40.

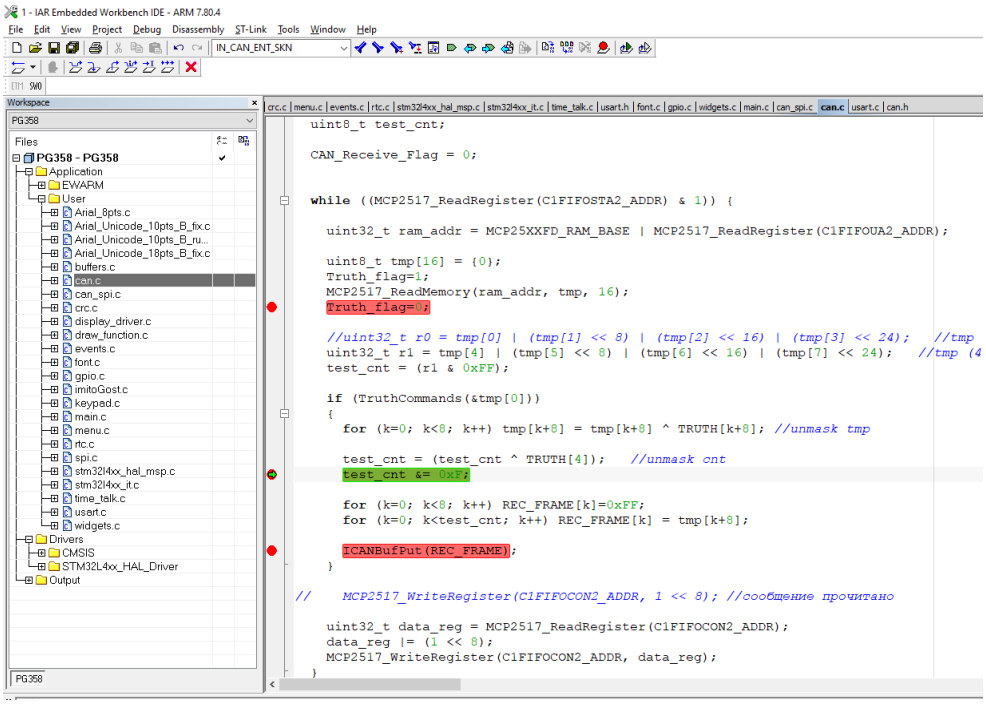


Рисунок 40 – Вторая точка остановки

Размаскированные данные представленны на рисунке 41.

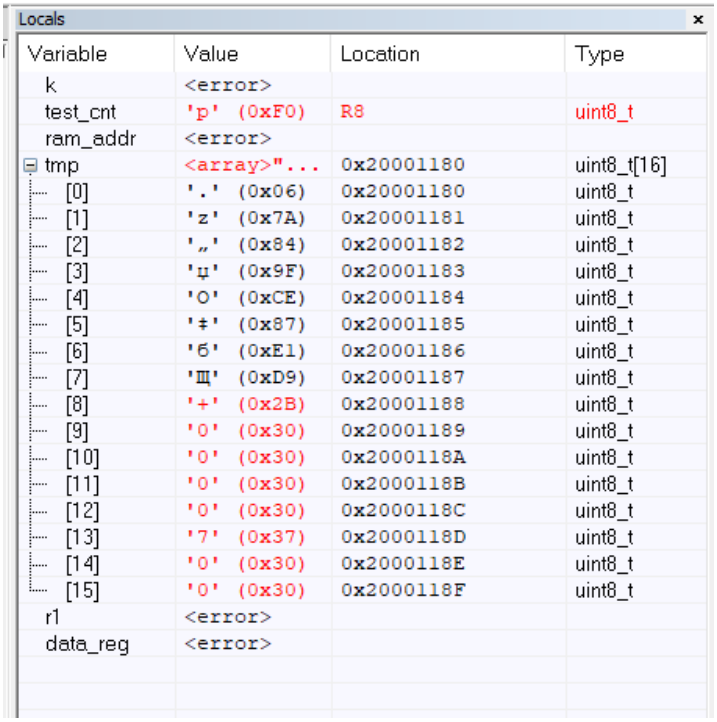


Рисунок 41 – Размаскированные данные

На следующей контрольной точке полученная команда записывается в массив для дальнейшей обработки (рисунок 42).

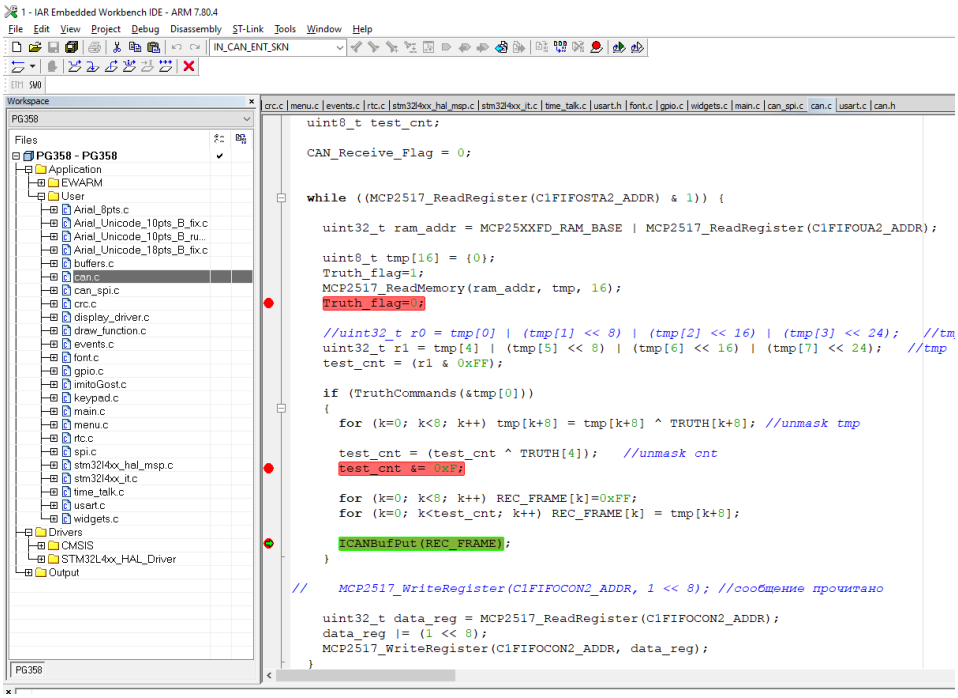


Рисунок 42 – Третья точка остановки

Принятая команда соответствует той, что была отправлена ранее (рисунок 43).

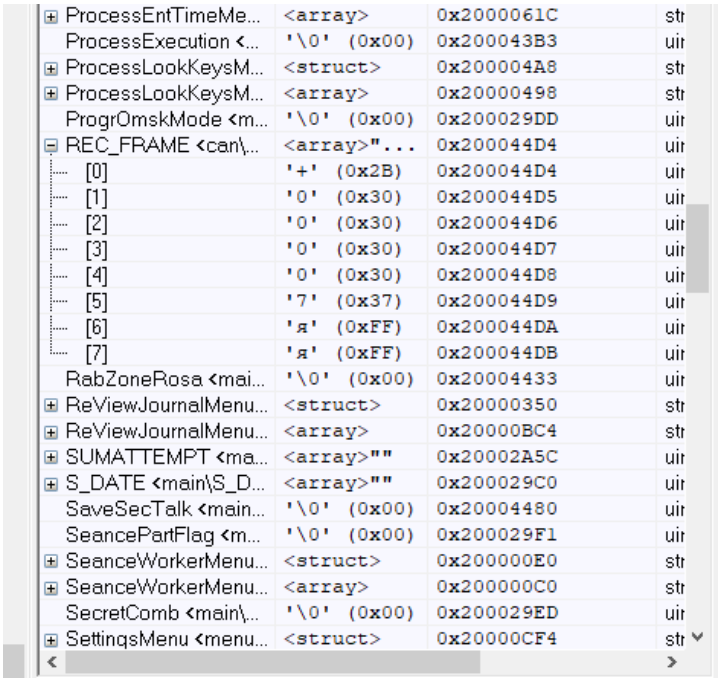


Рисунок 43 – Принятая команда

4 Охрана труда

4.1Вред здоровью при работе с ПЭВМ

Работая за компьютером, нужно помнить о неприятных последствиях для здоровья при несоблюдении мер безопасности. Проблемы со здоровьем

могут накапливаться со временем и здоровье будет постепенно ухудшаться, ниже представлены наиболее опасные факторы:

- Нагрузка на суставы. Долгая работа за компьютером в неправильной посадке, на суставы, а в особенности на кисти рук будет идти большая нагрузка, что может привести к хронической боли и онемению рук;
- Нагрузка на спину. При том же неправильном положении осанки во время работы за компьютером, есть шанс на то, что появится искривление позвоночника, грыжа межпозвоночного диска;
- Нагрузка на органы зрения. Работая за компьютером, можно так увлечься что глаза будут меньше моргать и увлажнять глаз из-за чего могут возникнуть проблемы со зрением и такие синдромы как «красного глаза» и «сухого глаза».

Ниже приведены простые правила, который помогут избежать серьезных проблем со здоровьем:

- Соблюдение необходимого расстояния до монитора. Правильным расстоянием до дисплея является расстояние, при котором пользователь прямой рукой может коснуться указательным пальцем до верхнего края монитора;
- соблюдение оптимального угла экрана. Оптимальным углом является наклон монитора в 30 градусов;
- организация перерывов. Примерно каждый час необходимо делать разминку для органов зрения, а именно отвлечься от монитора и делать гимнастику для глаз: фокусировать внимание на дальние объекты и переводить фокус с одних предметов на другие, рисовать различные фигуры глазами. Также примерно каждые полтора–два часа необходимо выполнять зарядку;

4.2 Требования безопасности при использовании устройства, коммутирующего сетевое напряжение 220В

Для предотвращения серьезных инцидентов, таких как поражение электрическим током, пожары и повреждение оборудования, критически важно соблюдать требования безопасности. Надлежащая изоляция,

использование качественных компонентов, регулярное обслуживание и тестирование устройства – путь к минимизации рисков и обеспечению надежной и безопасной эксплуатации. Важно информировать пользователей о правилах безопасности и правильного пользования устройством.

Перечень правил техники безопасности:

- не оставлять устройство без присмотра на длительное время;
- не использовать электроприборы, которые работает от питания без заземления;
- все проводящие части устройства должны иметь надёжную изоляцию. Корпус должен обеспечивать защиту от прикосновения к элементам, которые могут быть под напряжением;
- важно регулярно проверять механизм на предмет повреждений, износа и соответствия нормам безопасности. К этому относится также тестирование изоляции;
- пользователь должен быть хорошо проинструктирован о правилах безопасного использования устройства, а также уметь оказывать первую помощь в экстренных случаях, таких как поражение электрическим током;
- Должны соблюдаться требования условий, соответствующих техническим характеристикам (температура, влажность, отсутствие агрессивных сред и т.д.).

4.3 Требования условий эксплуатации устройства управления электроприборами на основе микроконтроллера STM32

С учетом всех составляющих элементов и их технических условий работы была разработана собственные требования условий к эксплуатации самого прибора:

- Температура эксплуатации: от –40°C до +85°C;
- Влажность: от 5% до 85% без конденсации;
- Диапазон давления: от 150 мм рт. ст. до 850 мм рт. ст.;
- Напряжение питания платы: 5 В.

Заключение

В рамках данной работы была решена задача по повышению надежности системы в части реализации маскирования входящих сообщений с целью защиты от возможной активации закладки в контроллере иностранного производителя с помощью внешней команды. Были изучены интерфейсы SPI и CAN, проведен анализ рынка нужных микросхем преобразователей SPI-CAN, было выбрано лучшее решение. Во время выполнения данной работы была доработана плата. Реализован узел контроллера CAN, включающий в себя микросхему преобразователь SPI-CAN и микросхему логического элемента XOR. Было написано ПО для взаимодействия с новым узлом, с помощью которого настраивается микросхема MCP2517FD. Для получения маски реализовано взаимодействия с СБ. После окончания разработки были проведены тесты с использованием эмулятора CAN – трансивера и ПО к нему.

Список использованных источников

1. Microchip Technology Inc. AN2606: MCP2517FD CAN FD Controller [Электронный ресурс] / Режим доступа: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP2517FD-External-CAN-FD-Controller-with-SPI-Interface-20005688B.pdf, свободный (дата обращения: 20.04.2026)
2. Microchip Technology Inc. *MCP2517FD Family Reference Manual* [Электронный ресурс] / Режим доступа: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP251XXFD-CAN-FD-Controller-Module-Family-Reference-Manual-20005678B.pdf, свободный (дата обращения: 20.04.2026)
3. STM32 Libraries [Электронный ресурс] / Режим доступа: https://www.compilenrun.com/docs/iot/stm32/stm32-development-environment/stm32-libraries/?utm_source, свободный. (дата обращения: 10.03.2026)
4. Noviello C. *Mastering STM32*. – Leanpub, 2018. – 520c.
5. STMicroelectronics. *Description of STM32L4 HAL and Low Layer Drivers (UM1884)* [Электронный ресурс] / Режим доступа: https://www.st.com/resource/en/user_manual/um1884-description-of-stm32l4l4-hal-and-lowlayer-drivers-stmicroelectronics.pdf, свободный (дата обращения: 04.03.2026).
6. Техническое руководство STM32 CMSIS [Электронный ресурс] / Режим доступа: https://www.keil.arm.com/cmsis ,свободный. (дата обращения: 10.03.2026)
7. Техническое описание STM32L476xx Reference Manual. [Электронный ресурс] / Режим доступа: https://www.st.com/resource/en/reference_manual/rm0432-stm32l4-series-advanced-armbased-32bit-mcus-stmicroelectronics.pdf, свободный. (дата обращения: 20.04.2026)
8. IAR Systems. *Getting Started with IAR Embedded Workbench for ARM* [Электронный ресурс] / Режим доступа:

https://www.iarsys.co.jp/download/LMS2/arm/7304/ewarm7304doc/common/doc/EW_GettingStarted.ENU.pdf , свободный (дата обращения: 20.03.2026)

Приложение А

(обязательное)

Преобразователь логических сигналов между шинами SPI и CAN

UML-диаграммы

Диаграмма общей структуры устройства

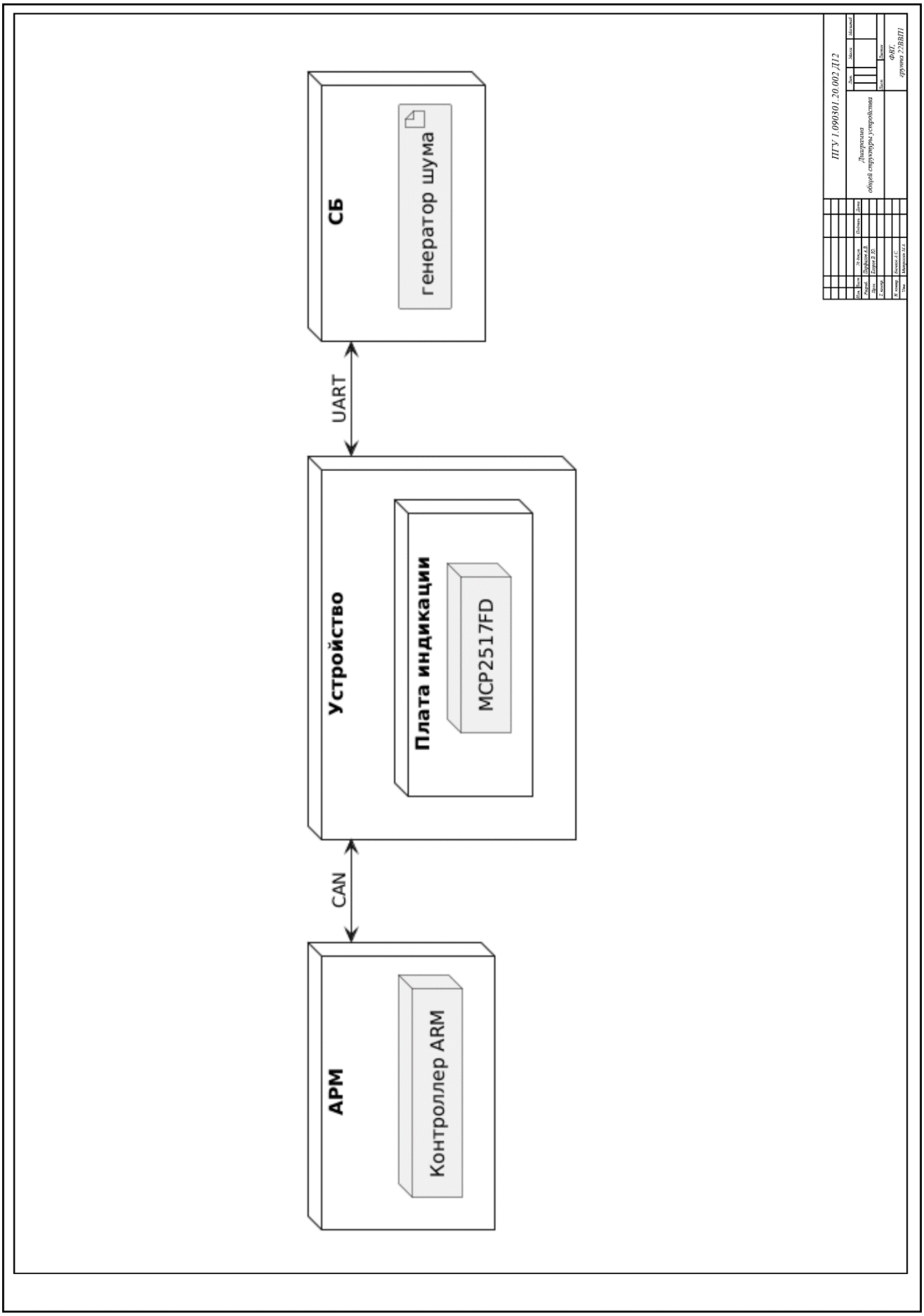
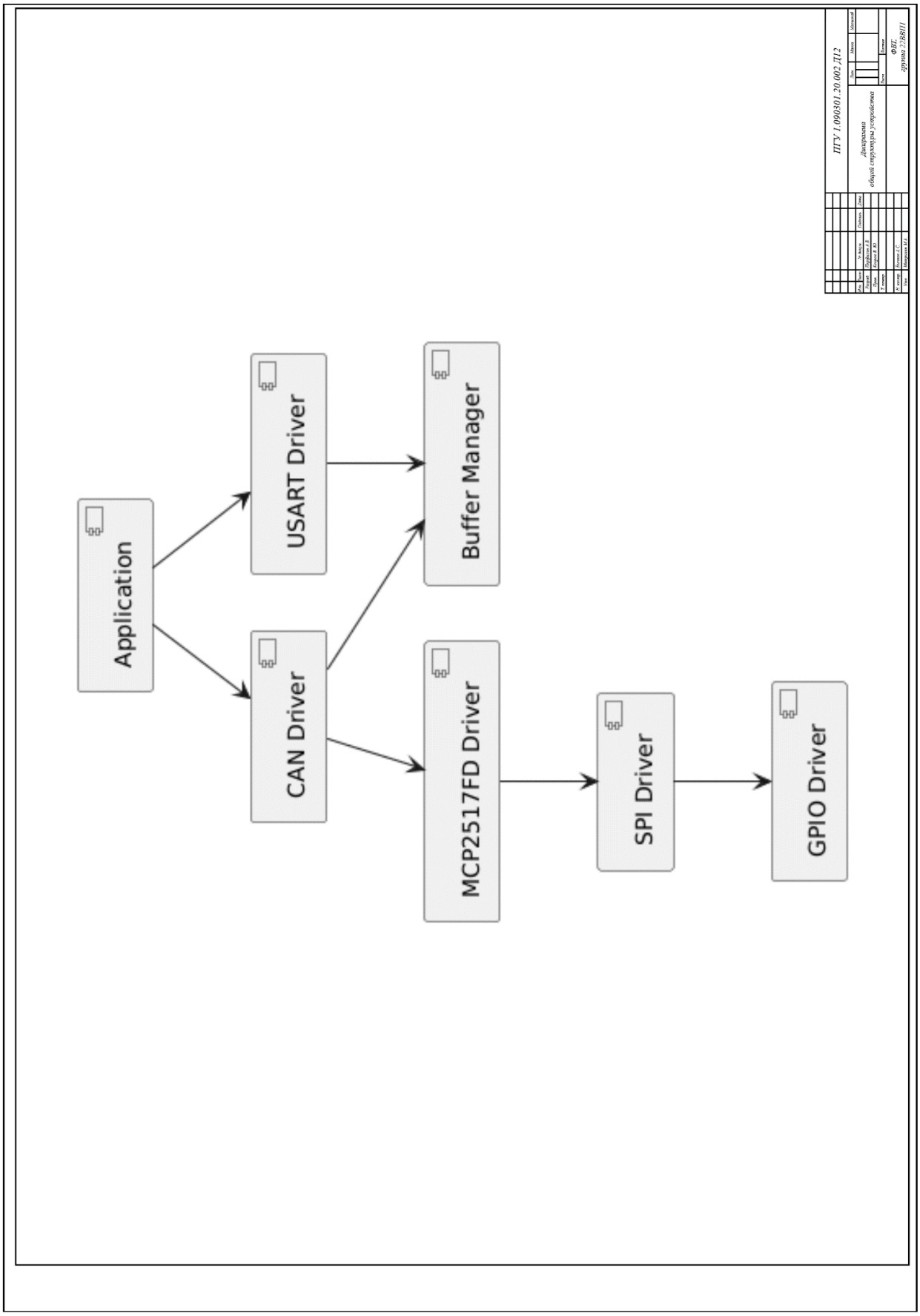


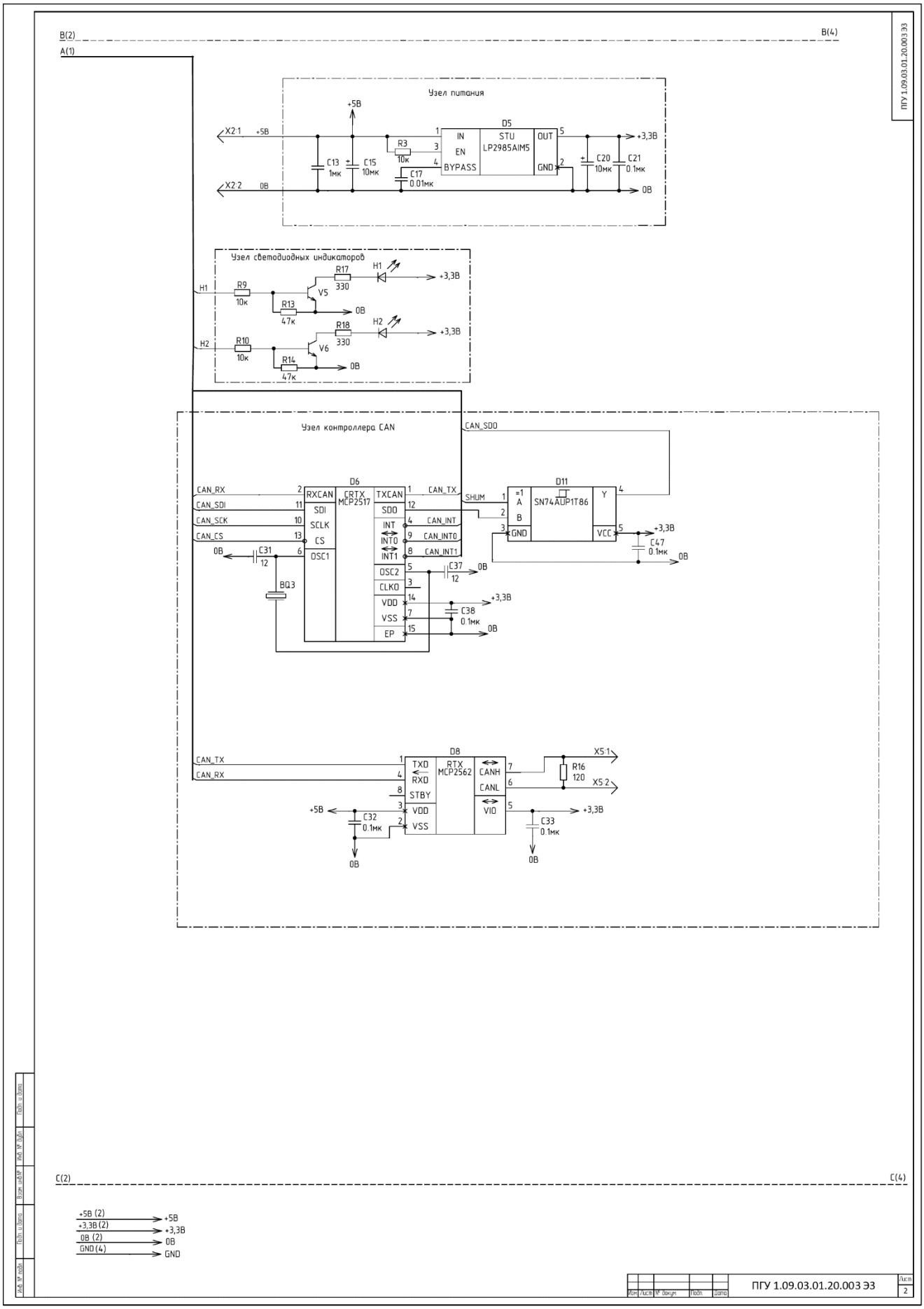
Диаграмма компонентов

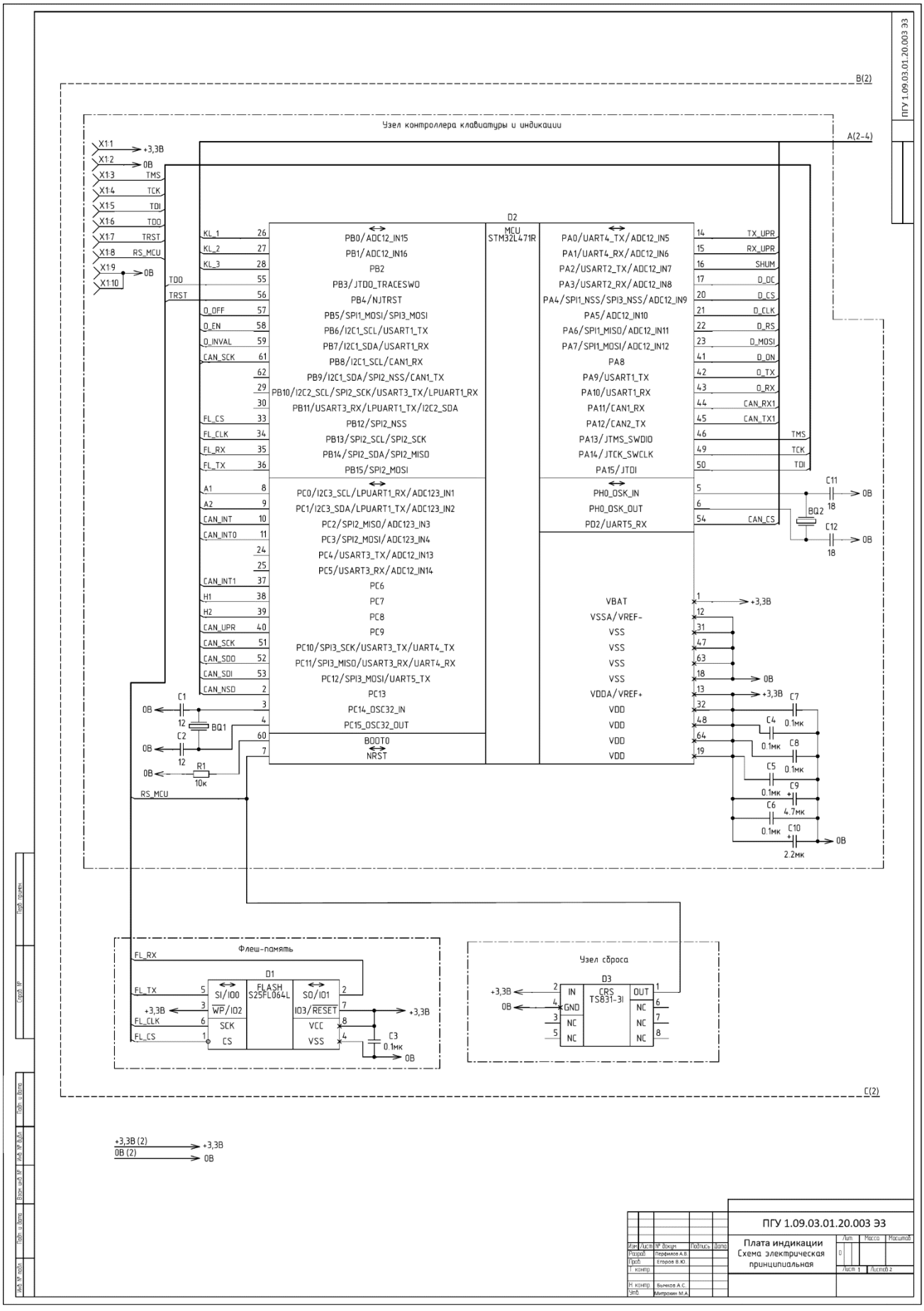


Приложение Б

(обязательное)

Схема электрическая принципиальная


B(2)
B(4)
A(1)
Узел питания
+5В
X2:1
+5В
X2:2
0В
D5
IN
EN
BYPASS
STU
LP2985AIM5
OUT
GND
R3
10к
C13
1мк
C15
10мк
C17
0.01мк
C20
10мк
C21
0.1мк
+3,3В
0В
Узел светодиодных индикаторов
H1
H2
R9
10к
R10
10к
R13
47к
R14
47к
R17
330
R18
330
V5
V6
Узел контроллера CAN
CAN_SDO
D6
RXCAN
CRTX
TXCAN
SDI
SDO
SCLK
CS
OSC1
INT
INT0
INT1
OSC2
CLKO
VDD
VSS
EP
MCP2517
CAN_RX
CAN_SDI
CAN_SCK
CAN_CS
CAN_TX
CAN_INT
CAN_INT0
CAN_INT1
C31
12
BQ3
C37
12
C38
0.1мк
SHUM
D11
=1
A
B
GND
Y
VCC
SN74AUP1T86
C47
0.1мк
D8
TXD
RXD
STBY
VDD
VSS
RTX
MCP2562
CANH
CANL
VIO
X5:1
X5:2
R16
120
C32
0.1мк
C33
0.1мк
C(2)
C(4)
+5В (2)
+3,3В (2)
0В (2)
GND (4)
+5В
+3,3В
0В
GND
ПГУ 1.09.03.01.20.003 ЭЗ
Лист
2


ПГУ 1.09.03.01.20.003 ЭЗ
B(2)
A(2-4)
C(2)
Узел контроллера клавиатуры и индикации
D2
MCU
STM32L471R
X1:1 +3,3В
X1:2 0В
X1:3 TMS
X1:4 TCK
X1:5 TDI
X1:6 TDO
X1:7 TRST
X1:8 RS_MCU
X1:9 0В
X1:10
TDO
TRST
KL_1 26
KL_2 27
KL_3 28
O_OFF 57
O_EN 58
O_INVAL 59
CAN_SCK 61
FL_CS 33
FL_CLK 34
FL_RX 35
FL_TX 36
A1 8
A2 9
CAN_INT 10
CAN_INT0 11
CAN_INT1 37
H1 38
H2 39
CAN_UPR 40
CAN_SCK 51
CAN_SDO 52
CAN_SDI 53
CAN_NSD 2
PB0/ADC12_IN15
PB1/ADC12_IN16
PB2
PB3/JTDO_TRACESWO
PB4/NJTRST
PB5/SPI1_MOSI/SPI3_MOSI
PB6/I2C1_SCL/USART1_TX
PB7/I2C1_SDA/USART1_RX
PB8/I2C1_SCL/CAN1_RX
PB9/I2C1_SDA/SPI2_NSS/CAN1_TX
PB10/I2C2_SCL/SPI2_SCK/USART3_TX/LPUART1_RX
PB11/USART3_RX/LPUART1_TX/I2C2_SDA
PB12/SPI2_NSS
PB13/SPI2_SCL/SPI2_SCK
PB14/SPI2_SDA/SPI2_MISO
PB15/SPI2_MOSI
PC0/I2C3_SCL/LPUART1_RX/ADC123_IN1
PC1/I2C3_SDA/LPUART1_TX/ADC123_IN2
PC2/SPI2_MISO/ADC123_IN3
PC3/SPI2_MOSI/ADC123_IN4
PC4/USART3_TX/ADC12_IN13
PC5/USART3_RX/ADC12_IN14
PC6
PC7
PC8
PC9
PC10/SPI3_SCK/USART3_TX/UART4_TX
PC11/SPI3_MISO/USART3_RX/UART4_RX
PC12/SPI3_MOSI/UART5_TX
PC13
PC14_OSC32_IN
PC15_OSC32_OUT
BOOT0
NRST
PA0/UART4_TX/ADC12_IN5
PA1/UART4_RX/ADC12_IN6
PA2/USART2_TX/ADC12_IN7
PA3/USART2_RX/ADC12_IN8
PA4/SPI1_NSS/SPI3_NSS/ADC12_IN9
PA5/ADC12_IN10
PA6/SPI1_MISO/ADC12_IN11
PA7/SPI1_MOSI/ADC12_IN12
PA8
PA9/USART1_TX
PA10/USART1_RX
PA11/CAN1_RX
PA12/CAN2_TX
PA13/JTMS_SWDIO
PA14/JTCK_SWCLK
PA15/JTDI
PH0_OSK_IN
PH0_OSK_OUT
PD2/UART5_RX
VBAT
VSSA/VREF-
VSS
VSS
VSS
VSS
VDDA/VREF+
VDD
VDD
VDD
VDD
14 TX_UPR
15 RX_UPR
16 SHUM
17 D_DC
20 D_CS
21 D_CLK
22 D_RS
23 D_MOSI
41 D_ON
42 D_TX
43 D_RX
44 CAN_RX1
45 CAN_TX1
46 TMS
49 TCK
50 TDI
5
6
54 CAN_CS
BQ2
C11 18 0В
C12 18 0В
1 +3,3В
12
31
47
63
18 0В
13 +3,3В
32
48
64
19
C7 0.1мк
C4 0.1мк
C8 0.1мк
C5 0.1мк
C9 0.1мк +
C6 4.7мк
C10 0.1мк + 2.2мк 0В
C1 0В 12
BQ1
C2 0В 12
R1 0В 10к
RS_MCU
Флеш-память
D1
FLASH
S25FL064L
FL_RX
FL_TX 5
+3,3В 3
FL_CLK 6
FL_CS 1
SI/IO0
WP/IO2
SCK
CS
SO/IO1 2
IO3/RESET 7
VCC 8
VSS 4
+3,3В
C3 0.1мк 0В
Узел сброса
D3
CRS
TS831-3I
+3,3В 2
0В 4
3
5
IN
GND
NC
NC
OUT 1
NC 6
NC 7
NC 8
+3,3В (2) +3,3В
0В (2) 0В
ПГУ 1.09.03.01.20.003 ЭЗ
Плата индикации
Схема электрическая принципиальная
Разраб. Перфилов А.В.
Пров. Егоров В.Ю.
Т. контр.
Н. контр. Бычков А.С.
Утв. Митрофин М.А.
Лит. Масса Масштаб
Лист 1 Листов 2

Приложение В

(обязательное)

Листинг программы

```
int main(void)
{
  // Reset of all peripherals, Initializes the Flash interface
and the Systick.
  HAL_Init();

  SystemClock_Config();   // Configure the system clock and the
Systick.

  // Initialize all configured peripherals
  MX_GPIO_Init();

  D_ON_GPIO_Low_state();
//  Can_Stby_Off(); //управление теперь после чтения параметров

  Red_Off();        //Инициализация светодиодов. Состояние - Выкл
  Green_Off();      //Инициализация светодиодов. Состояние - Выкл
  sFlash_CS_High();
  MX_SPI2_Init();  //SPI для работы с внешней флеш

  MX_CRC_Init();

  if (COMMAND_PORT)
  {
    EX_MAX3221_DeInit();
    MX_UART4_Init();
  }
  else
  {
    MX_USART1_UART_Init();      //UART для отладки
    EX_MAX3221_Init();
  }

  MX_RTC_Init();
//-----------------------------------------------------------------


  InitKeypad();    // клавиатура

  // Infinite loop
  while (1)
  {
    ResetParam();

    PreStartingControl();      //предпусковой контроль

    while (1)
    {
      if (CE_pointer) ConnectEvents();
      if (Eventpointer) ProcessEvents();

      if (OCANBuf.Cnt) SendCommCAN();   // передача команд по CAN
интерфейсу
      if (ICANBuf.PtrW!=ICANBuf.PtrR) TestCommCAN();  // анализ
команд по CAN интерфейсу
```

```
            if (OmspBuf.Cnt) SendComm();   // передача команд в CE171

            if (!GameOver)
            {
              if (((CAN_StateNoise_Flag_1 != 3)||(CAN_StateNoise_Flag_2
!= 3))&&(!CAN_WaitAnswer_Flag)) InquiryCanNoise();

              if (CAN_Receive_Flag) ReceiveCanData();

              if (ImspBuf.PtrW!=ImspBuf.PtrR) TestComm();   // анализ
команд от CE171
              if ((DrawKosh)&&(!KoshPullTimer)) DrawPullKosh();
// отображение "Кош пороговый"

              if (EnableDrawCursor) BlinkCursor();
              if ((EnableTimeTalk)&&(TIME_TALK[7]!=SaveSecTalk))
SolveTimeTalk();
              if (!TimeTimer) InitGetTime();
              if (!StatusTimer) InitGetStatus();
              if ((StatusTimer<5005)&&(StatusTimer>5000))
              {
                StatusTimer = 3000;
                TestRezultStatus();
              }

      //-----
              if (!ButTimer) TestKeypad();
              if (!NsdUprTimer) TestNsdUpr();
              if ((DisplayMode==1)&&(!DisplayTimer))
              {
                DisplayMode++;
                DisplaySleepMode();
              }

              if ((MenuMode)&&(!MenuTimer))
              {
                MenuTimer=30000;
                if (!Get_Exclusion_Menu())
                {
                  if (ControlPersonalMenu) ExitFromSpecMenuFunc();
                  else ExitFromMenuFunc();
                  if ((!CAN_Disable)&&(DoorUVK_State)&&(DoorDNI_State))
Can_Stby_Off();    //ON CAN
                }
              }
              if ((SvDiodMode>1)&&(!SvDiodTimer)) BlinkSvDiod();
            }

            if (CurrState) DrawCurrState();

      //-------------------------------------------

      //-------------------------------------------

          }
        }
      }
      //
```

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

          // Configure LSE Drive Capability
        //
        HAL_PWR_EnableBkUpAccess();
        __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
        // 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 = 6;
        RCC_OscInitStruct.PLL.PLLN = 64;
        RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
        RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
        RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV4;
        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_DIV1;
        RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

        if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) !=
HAL_OK)
        {
          Error_Handler();
        }
        PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_RTC|RCC_PERIPHCLK_USART1

|RCC_PERIPHCLK_UART4|RCC_PERIPHCLK_UART5;
        PeriphClkInit.Usart1ClockSelection =
RCC_USART1CLKSOURCE_SYSCLK;
        PeriphClkInit.Uart4ClockSelection = RCC_UART4CLKSOURCE_SYSCLK;
        PeriphClkInit.Uart5ClockSelection = RCC_UART5CLKSOURCE_SYSCLK;
        PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSE;
        if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
        {
          Error_Handler();
        }

        // Configure the main internal regulator output voltage
```

```
        if
(HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) !=
HAL_OK)
        {
          Error_Handler();
        }

        // Configure the Systick interrupt time
        HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

        // Configure the Systick
        HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

        // SysTick_IRQn interrupt configuration
        HAL_NVIC_SetPriority(SysTick_IRQn, 1, 0);
      }

      //---------------------------------------------------------------
---------------

      void Delay(uint32_t nTime)
      {
        Sys_TimingDelay = nTime;

        while(Sys_TimingDelay)
        {
          asm(" nop ");
        }
      }

      //---------------------------------------------------------------
---------------


      //
        // @brief  This function is executed in case of error
occurrence.
        // @retval None
      //
      void Error_Handler(void)
      {
        // User can add his own implementation to report the HAL error
return state
        __disable_irq();
        while (1)
        {
          asm(" nop ");
        }
      }

      #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 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)
      }
      #endif // USE_FULL_ASSERT

      /************************ END OF FILE
****************************************/
```

Файл can.c

```
      // Includes ------------------------------------------------------
---------------

      #include "can.h"
      #include "main.h"
      #include "usart.h"
      #include "events.h"

      #include "draw_function.h"
      #include "font.h"
      #include "display_driver.h"

      #include "time_talk.h"
      #include "spi.h"
      #include "buffers.h"
      #include "can_spi.h"

      uint8_t ICANBUF_[ICANBUFLEN][8];
      uint8_t OCANBUF_[OCANBUFLEN][8];

      CANBuf ICANBuf;
      CANBuf OCANBuf;

      void ICANBufInit(uint16_t Len);
      void OCANBufInit(uint16_t Len);
      void ICANBufPut(uint8_t* data);
      void OCANBufPut(uint8_t* data);
      void ICANBufGet(uint8_t* data);
      void OCANBufGet(uint8_t* data);
      void TestCommCAN(void);
      void SendCommCAN(void);

      uint32_t pTxMailbox;

      uint8_t REC_FRAME[8];

      void ReceiveCanData(void);

      uint8_t TestErrEWGF=0;
      uint8_t TestErrEPVF=0;
      uint8_t TestErrBOFF=0;

      uint8_t TestErrOverrun=0;
      uint8_t TestErrFull=0;

      //uint8_t SEND_FRAME[8];

      //------------------------------------------------------------------
---------------
      void ICANBufInit(uint16_t Len)
      {
        uint16_t i,j;
```

```
  ICANBuf.PtrR=0;
  ICANBuf.PtrW=0;
  ICANBuf.Cnt=0;
  ICANBuf.Len=Len;

  for(i=0;i<Len;i++)
  {
    for(j=0;j<8;j++)
    {
      ICANBUF_[i][j]=0;
    }
  }
}
//-----------------------------------------------------------------
---------------
void OCANBufInit(uint16_t Len)
{
  uint16_t i,j;

  OCANBuf.PtrR=0;
  OCANBuf.PtrW=0;
  OCANBuf.Cnt=0;
  OCANBuf.Len=Len;

  for(i=0;i<Len;i++)
  {
    for(j=0;j<8;j++)
    {
      OCANBUF_[i][j]=0;
    }
  }
}
//-------------------- Запись в буфер ------------------
void ICANBufPut(uint8_t* data)
{
  for (uint8_t i=0; i<8; i++)
  {
    ICANBUF_[ICANBuf.PtrW][i] = data[i];
  }
  ICANBuf.PtrW++;
  ICANBuf.Cnt++;
  if(ICANBuf.PtrW>=ICANBuf.Len) ICANBuf.PtrW=0;
}
//-------------------- Запись в буфер ------------------
void OCANBufPut(uint8_t* data)
{
  for (uint8_t i=0; i<8; i++)
  {
    OCANBUF_[OCANBuf.PtrW][i] = data[i];
  }
  OCANBuf.PtrW++;
  OCANBuf.Cnt++;
  if(OCANBuf.PtrW>=OCANBuf.Len) OCANBuf.PtrW=0;
}

//-------------------- Чтение из буфера ----------------
void ICANBufGet(uint8_t* data)
{
  for (uint8_t i=0; i<8; i++)
  {
    data[i] = ICANBUF_[ICANBuf.PtrR][i];
  }
  ICANBuf.PtrR++;
```

```
        ICANBuf.Cnt--;
        if(ICANBuf.PtrR==ICANBuf.Len) ICANBuf.PtrR=0; // если указатель
чтения дошёл до последнего элемента буфера, указатель чтения
сбрасывается в ноль.
      }
      //-------------------- Чтение из буфера ----------------
      void OCANBufGet(uint8_t* data)
      {
        for (uint8_t i=0; i<8; i++)
        {
          data[i] = OCANBUF_[OCANBuf.PtrR][i];
        }
        OCANBuf.PtrR++;
        OCANBuf.Cnt--;
        if(OCANBuf.PtrR==OCANBuf.Len) OCANBuf.PtrR=0; // если указатель
чтения дошёл до последнего элемента буфера, указатель чтения
сбрасывается в ноль.
      }

      //-------------------------------------------------------------
---------------
      void TestCommCAN(void)
      {
        uint8_t k;

        if (!UPR_State)
        {
          UPR_State=1;   //status UPR - yes connect
          if (GetValidScreen()) CurrState=1;
        }

        ICANBufGet(TEST_FRAME);

        if ((CAN_Disable)||(GameOver)) return;

        switch (TEST_FRAME[0])
        {
          //-------------------------------------------------------
---------------
          case IN_CAN_OUT_CALL:
            if (!Connect)
            {
              if (!GetValidConnect())
              {
                for (k=0; k<8; k++) TEST_FRAME[k]=0xff;
                TEST_FRAME[0]=OUT_CAN_OTBOY;
                OCANBufPut(TEST_FRAME);
              }
              else
              {
                HaveNewMissingCall=0;
                ResetConnectParam();
                Connect = 1;  //исх вызов в ручном режиме от Веги
                //SvDiodSetState();
                InitFormConnect();
              }
            }
            else
            {
              for (k=0; k<8; k++) TEST_FRAME[k]=0xff;
              TEST_FRAME[0]=OUT_CAN_OTBOY;
              OCANBufPut(TEST_FRAME);
            }
```

```
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_OTBOY:
            if (Connect) PressOtboy=1;
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_ANS_CALL:
            Vega_AnsOnIncall=1;
            StartTimeTalk();    //запуск таймера разговора
            DrawConnectState(CE_pointer);
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_INQ_STATE_AC:
            for (k=0; k<8; k++) TEST_FRAME[k]=0xff;
            TEST_FRAME[0]=OUT_CAN_STATE_AC;

            if (ModeARS) TEST_FRAME[1]=OUT_CAN_AC_ERR;
            else if (!(KAN_State&1)) TEST_FRAME[1]=OUT_CAN_AC_ERR;
            else TEST_FRAME[1]=0;

            if (!(KAN_State&4)) TEST_FRAME[2]=OUT_CAN_CHANNEL_BUSY;
            else TEST_FRAME[2]=OUT_CAN_CHANNEL_FREE;
            if (!(KAN_State&2)) TEST_FRAME[3]=OUT_CAN_CHANNEL_NOREADY;
            else TEST_FRAME[3]=OUT_CAN_CHANNEL_READY;
            TEST_FRAME[4]=((KAN_State>>3)&7);
            OCANBufPut(TEST_FRAME);
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_SHLF_2:
                        // отбить соединение
        //     ResetConnectParam();
            if (Connect) {
              EnableTimeTalk=0;
              PressOtboy=1;
            }
            else {
            if (GetValidScreen()) CurrState=1;
            }


            FormPacketKK(OUT_KK_SET_SHLF_2, TEMP_OUT_PACKET,
OUT_PACKET, 0);    // запрос КИ
            ModeWork=2;
            FlashBuff[4]=ModeWork;
            SaveConfig();
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_RAB_AC:
                  // отбить соединение
        //     ResetConnectParam();
            if (Connect) {
              EnableTimeTalk=0;
              PressOtboy=1;
            }
            else {
            if (GetValidScreen()) CurrState=0;
            }
```

```
            FormPacketKK(OUT_KK_SET_RAB, TEMP_OUT_PACKET, OUT_PACKET,
0);
            ModeWork=0;
            FlashBuff[4]=ModeWork;
            SaveConfig();
            CAN_Control_AC = 0;
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_APC_ON_OFF:
            if (TEST_FRAME[1])
            {
              FormPacketKK(OUT_KK_ARS_ON, TEMP_OUT_PACKET, OUT_PACKET,
0);
              TempModeARS=0;
            }
            else
            {
              FormPacketKK(OUT_KK_ARS_OFF, TEMP_OUT_PACKET, OUT_PACKET,
0);
              TempModeARS=1;
            }
            ModeARS=TempModeARS;
            FlashBuff[5]=ModeARS;
            SaveConfig();
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_INQ_STATE_CA:
            for (k=0; k<8; k++) TEST_FRAME[k]=0xff;
            TEST_FRAME[0]=OUT_CAN_STATE_CA;
            if (!GetValidStateCA()) TEST_FRAME[1]=2;
            else TEST_FRAME[1]=0;

            if (CAN_Control_CA) TEST_FRAME[2]=4;
            else if (CAN_Control_AC) TEST_FRAME[2]=5;
            else TEST_FRAME[2]=ModeWork;

            if (!Connect) TEST_FRAME[3]=0;
            else
            {
              if ((CE_pointer==3)||(CE_pointer==4)||(CE_pointer==5))
              {
                if (Connect==1) TEST_FRAME[3]=2;
                else TEST_FRAME[3]=1;
              }
              else if
((CE_pointer==6)||(CE_pointer==7)||(CE_pointer==8))
              {
                TEST_FRAME[3]=3;
              }
              else TEST_FRAME[3]=4;
            }

            if (!GetStateKeys()) TEST_FRAME[4]=1;
            else TEST_FRAME[4]=0;

      //      if (((KAN_State&1))&&((KAN_State&2))&&((KAN_State&4)))
TEST_FRAME[5]=0;
            if (((!ModeARS)&&((KAN_State&1))&&((KAN_State&4)))
TEST_FRAME[5]=0;
            else TEST_FRAME[5]=1;
            OCANBufPut(TEST_FRAME);
```

```
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_INQ_STATE_KEYS:
            for (k=0; k<8; k++) TEST_FRAME[k]=0xff;
            TEST_FRAME[0]=OUT_CAN_STATE_KEYS;
            if (KeyNSD!=3) TEST_FRAME[1]=0;
            else TEST_FRAME[1]=1;
            if (RabZoneRosa==0) TEST_FRAME[2]=0x20;
            else if (RabZoneRosa==1) TEST_FRAME[2]=0x30;
            else TEST_FRAME[2]=0x31;
            if (KeyRosaZ1==4) TEST_FRAME[3]=1;
            else TEST_FRAME[3]=0;
            if (KeyRosaZ2==4) TEST_FRAME[4]=1;
            else TEST_FRAME[4]=0;
            OCANBufPut(TEST_FRAME);
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_RESET_KEYS:
            TEMP_OUT_PACKET[0]=1;       // value
            FormPacketKK(OUT_KK_EXTR_DELKEYS, TEMP_OUT_PACKET,
OUT_PACKET, 1);             // экстр стирание CAN

            if (Connect) {
              EnableTimeTalk=0;
              PressOtboy=1;
            }
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_SMENA_KEYS:
            TEMP_OUT_PACKET[0]=1;       // value
            FormPacketKK(OUT_KK_SMENA_KEY, TEMP_OUT_PACKET, OUT_PACKET,
1);         // смена КИ CAN
            if (Connect) {
              EnableTimeTalk=0;
              PressOtboy=1;
            }
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_SHLF_1:

                  // отбить соединение
        //      ResetConnectParam();
            if (Connect) {
              EnableTimeTalk=0;
              PressOtboy=1;
            }
            else {
            if (GetValidScreen()) CurrState=1;
            }


            FormPacketKK(OUT_KK_SET_SHLF_1, TEMP_OUT_PACKET,
OUT_PACKET, 0);
            ModeWork=1;
            FlashBuff[4]=ModeWork;
            SaveConfig();
            break;
```

```
          //-------------------------------------------------------------
---------------
          case IN_CAN_SHLF_3:
                  // отбить соединение
        //     ResetConnectParam();
            if (Connect) {
              EnableTimeTalk=0;
              PressOtboy=1;
            }
            else {
            if (GetValidScreen()) CurrState=1;
            }

              FormPacketKK(OUT_KK_SET_SHLF_3, TEMP_OUT_PACKET,
OUT_PACKET, 0);
              ModeWork=3;
              FlashBuff[4]=ModeWork;
              SaveConfig();

              break;
            //-------------------------------------------------------------
---------------
          case IN_CAN_MKF_ON:
              FormPacketKK(OUT_KK_MKF_ON, TEMP_OUT_PACKET, OUT_PACKET,
0);
              break;
            //-------------------------------------------------------------
---------------
          case IN_CAN_MKF_OFF:
              FormPacketKK(OUT_KK_MKF_OFF, TEMP_OUT_PACKET, OUT_PACKET,
0);
              break;
            //-------------------------------------------------------------
---------------
          case IN_CAN_VOLUME_MKF:
              if (!TEST_FRAME[1]) TEST_FRAME[1] = 1;
              VolumeMkfLevel = TEST_FRAME[1];
              FlashBuff[3]=VolumeMkfLevel;
              SaveConfig();

              if (!SpeechMode)
              {
                TEMP_OUT_PACKET[0]=VolumeMkfLevel;       // value
                FormPacketKK(OUT_KK_VOLUME_MKF, TEMP_OUT_PACKET,
OUT_PACKET, 1);       // громкость мкф
              }
              break;
            //-------------------------------------------------------------
---------------
          case IN_CAN_VOLUME_TLF:
              if (!TEST_FRAME[1]) TEST_FRAME[1] = 1;
              VolumeSpeechLevel = TEST_FRAME[1];
              FlashBuff[2]=VolumeSpeechLevel;
              SaveConfig();
              if (!SpeechMode)
              {
                TEMP_OUT_PACKET[0]=VolumeSpeechLevel;       // value
                FormPacketKK(OUT_KK_VOLUME_TLF, TEMP_OUT_PACKET,
OUT_PACKET, 1);       // громкость т/ф
              }
              break;
---------------
```

```
          case IN_CAN_CONTROL_CA:
            if (ControlPersonalMenu)
            {
              Elaboration=0;
              BlockKeypad=0;
              ControlPersonalMenu=0;
              FormPacketKK(OUT_KK_EX_SPECMODE, TEMP_OUT_PACKET,
OUT_PACKET, 0);     // Выход из спец меню
            }
            EnableDrawCursor=0;
            Eventpointer=0;
            ProcessExecution=0;
            MenuMode=0;

            // отбить соединение
       //     ResetConnectParam();
            if (Connect) {
              EnableTimeTalk=0;
              PressOtboy=1;
            }
            else {
              CurrState=1;
            }

            CAN_Control_CA = 1;
            for (k=0; k<8; k++) TEST_FRAME[k]=0xff;
            if (!GetValidStateCA()) TEST_FRAME[0]=OUT_CAN_GEAR_AVR;
            else TEST_FRAME[0]=OUT_CAN_GEAR_NORM;
            OCANBufPut(TEST_FRAME);
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_RABOTA_CA:
            CAN_Control_CA = 0;

                  // отбить соединение
       //     ResetConnectParam();
            if (Connect) {
              EnableTimeTalk=0;
              PressOtboy=1;
            }
            else {
              CurrState=1;
            }

            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_CONTROL_AC:
            if (ControlPersonalMenu)
            {
              Elaboration=0;
              BlockKeypad=0;
              ControlPersonalMenu=0;
              FormPacketKK(OUT_KK_EX_SPECMODE, TEMP_OUT_PACKET,
OUT_PACKET, 0);     // Выход из спец меню
            }
            EnableDrawCursor=0;
            Eventpointer=0;
            ProcessExecution=0;
            MenuMode=0;
```

```
                // отбить соединение
      //     ResetConnectParam();
            if (Connect) {
              EnableTimeTalk=0;
              PressOtboy=1;
            }
            else {
              CurrState=1;
            }

            CAN_Control_AC = 1;

            if (ModeARS)
            {
             State_AC_for_CAN=2;
             for (k=0; k<8; k++) TEST_FRAME[k]=0xff;
             TEST_FRAME[0]=OUT_CAN_OMSK_AVR;
             OCANBufPut(TEST_FRAME);

             LCD_PutStrig(TEXT_STR3_2, 16, 0, "АС неисправна");
             RefreshDisplay();
            }
            else
            {
              State_AC_for_CAN=0;
              FormPacketKK(OUT_KK_CONTROL_OMSK, TEMP_OUT_PACKET,
OUT_PACKET, 0);     // Контроль АРС Омск
            }
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_INQ_KN:
            if (Connect) {

              for (k=0; k<8; k++) TEST_FRAME[k]=0xff;
              TEST_FRAME[0]=OUT_CAN_CURR_KN;
              for(k=0; k<5; k++)
              {
                TEST_FRAME[k+1]=CURRKN[k];
              }
              OCANBufPut(TEST_FRAME);

            }
            else FormPacketKK(OUT_KK_INQ_KN, TEMP_OUT_PACKET,
OUT_PACKET, 0);
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_CONTROL_BAT:
            for (k=0; k<8; k++) TEST_FRAME[k]=0xff;
            TEST_FRAME[0]=OUT_CAN_LEV_BAT;
            TEST_FRAME[1]=Battery_State;
            OCANBufPut(TEST_FRAME);
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_PROGR_OMSK_MODE:
            if (TEST_FRAME[1])
            {
              FormPacketKK(OUT_KK_PROGR_OMSK_ON, TEMP_OUT_PACKET,
OUT_PACKET, 0);
              ProgrOmskMode=1;
```

```
              if (ControlPersonalMenu)
              {
                Elaboration=0;
                BlockKeypad=0;
                ControlPersonalMenu=0;
                FormPacketKK(OUT_KK_EX_SPECMODE, TEMP_OUT_PACKET,
OUT_PACKET, 0);          // Выход из спец меню
              }
              EnableDrawCursor=0;
              Eventpointer=0;
              ProcessExecution=0;
              MenuMode=0;

              // отбить соединение

       //     ResetConnectParam();
              if (Connect) {
                EnableTimeTalk=0;
                PressOtboy=1;
              }
              else {
              CurrState=1;
              }

            }
            else
            {
              FormPacketKK(OUT_KK_PROGR_OMSK_OFF, TEMP_OUT_PACKET,
OUT_PACKET, 0);

              ProgrOmskMode=0;
              CurrState=1;
            }
            break;
          //-------------------------------------------------------------
---------------
          case IN_CAN_DATA_PR_OMSKMODE:
            if (ProgrOmskMode)
            {
              for (k=1; k<8; k++)
              {
                if (TEST_FRAME[k]==0xFF) break;
                TEMP_OUT_PACKET[k-1] = TEST_FRAME[k];
              }
              if (k==8) k=7;
              FormPacketKK(OUT_KK_PR_OMSK_START, TEMP_OUT_PACKET,
OUT_PACKET, k);
            }
            break;

          //-------------------------------------------------------------
---------------
          case IN_CAN_DIG_PACK_1:
          case IN_CAN_DIG_PACK_2:
          case IN_CAN_DIG_PACK_3:
          case IN_CAN_DIG_PACK_4:
          case IN_CAN_DIG_PACK_5:
            if (TEST_FRAME[0]==IN_CAN_DIG_PACK_1)
            {
              for(k=0; k<34; k++) CAN_DIG_NUM[k]=0;
              CAN_CntDigNum=0;
            }
            for (k=1; k<8; k++)
```

```
            {
              if (TEST_FRAME[k]==0xFF) break;
              CAN_DIG_NUM[CAN_CntDigNum++]=TEST_FRAME[k];
            }
            break;
          //-----------------------------------------------------------
---------------
          case IN_CAN_DIG_PACK_END:
            CAN_DIG_NUM[CAN_CntDigNum++]=TEST_FRAME[1];
            CAN_DIG_NUM[CAN_CntDigNum++]=TEST_FRAME[2];
            for (k=0; k<CAN_CntDigNum; k++)
            {
              TEMP_OUT_PACKET[k] = CAN_DIG_NUM[k];
            }
            FormPacketKK(OUT_KK_DOP_NUM_START, TEMP_OUT_PACKET,
OUT_PACKET, CAN_CntDigNum);
            break;
          //-----------------------------------------------------------
---------------
          case IN_CAN_ENT_SKN:
            for(k=0; k<5; k++)
            {
              TEMP_OUT_PACKET[k] = TEST_FRAME[k+1];
            }
            FormPacketKK(OUT_KK_ENT_SKN_START, TEMP_OUT_PACKET,
OUT_PACKET, 5);
            break;

            default:
            break;
        }
      }

      //-----------------------------------------------------------------
---------------

      void SendCommCAN(void)
      {
        uint8_t SEND_FRAME[8];
        uint8_t i;
        uint8_t cnt_bytes;

        uint32_t ram_addr=0;
        uint32_t T0 = 0, T1 = 0, tmp = 0;

        uint8_t ctrl[16] = {0};

        if ((MCP2517_ReadRegister(C1FIFOSTA1_ADDR) & 1))  // FIFO is
not full
        {
          OCANBufGet(TEST_FRAME);

          for (i=0; i<8; i++) SEND_FRAME[i]=0;
          cnt_bytes=0;
          for (i=0; i<8; i++)
          {
            if (TEST_FRAME[i]!=0xff)
            {
              cnt_bytes++;
              SEND_FRAME[i] = TEST_FRAME[i];
            }
            else break;
          }
```

```
        if ((CAN_Disable)||(GameOver)) return;

       /*   pHeader.DLC = cnt_bytes;
          pHeader.IDE = CAN_ID_EXT;
          pHeader.RTR = CAN_RTR_DATA;
          pHeader.ExtId = 0x417300;

          if (HAL_CAN_AddTxMessage(&hcan1, &pHeader, SEND_FRAME,
&pTxMailbox)  != HAL_OK )
          {
            Error_Handler();
          }
          */

          ram_addr = MCP25XXFD_RAM_BASE |
MCP2517_ReadRegister(C1FIFOUA1_ADDR);

          T0 = (0x417300 & 0x1FFFFFFF);
          tmp |= (T0 & 0x3FFFF) << 11;
          tmp |= (T0 >> 18) & 0x7FF;

      //    T1 |= 0<<7;
      //    T1 |= 0<<6;
      //    T1 |= 0<<5;
          T1 |= 1<<4;
          T1 |= (cnt_bytes & 0xF);

           // write T0/T1 + payload
          uint8_t wb[16] = {0};
          wb[0] = (tmp >> 0) & 0xFF;
          wb[1] = (tmp >> 8) & 0xFF;
          wb[2] = (tmp >> 16) & 0xFF;
          wb[3] = (tmp >> 24) & 0xFF;
          wb[4] = (T1 >> 0) & 0xFF;
          wb[5] = (T1 >> 8) & 0xFF;
          wb[6] = (T1 >> 16) & 0xFF;
          wb[7] = (T1 >> 24) & 0xFF;

          uint16_t data_length = cnt_bytes;
          memcpy(&wb[8], SEND_FRAME, data_length);

          uint8_t SizePackage = RoundingToMul4(cnt_bytes + 8);
          MCP2517_WriteMemory((uint16_t)(ram_addr), wb, SizePackage);

      //----проверка передаваемой информации
          MCP2517_ReadMemory(ram_addr, ctrl, 16);

          for (i=0; i<(cnt_bytes+8); i++) {
            if (ctrl[i] != wb[i]) {
              for (i=0; i<cnt_bytes; i++) wb[i+8] = 0;
              SizePackage = RoundingToMul4(cnt_bytes + 8);
              MCP2517_WriteMemory((uint16_t)(ram_addr), wb,
SizePackage);
              return;
            }
          }
      //----

          uint32_t data_reg = MCP2517_ReadRegister(C1FIFOCON1_ADDR);
          data_reg |= ((1 << 8) | (1 << 9));  // Инкремент
```

запрос на передачу сообщения и увеличить указатель памяти для записи следующего сообщения

```
          MCP2517_WriteRegister(C1FIFOCON1_ADDR, data_reg);

        }
      }



      //-------------------------------------------------------------
---------------

      void HAL_CAN_MspInit(CAN_HandleTypeDef* canHandle)
      {
        GPIO_InitTypeDef GPIO_InitStruct = {0};
        if(canHandle->Instance==CAN1)
        {
          // CAN1 clock enable
          __HAL_RCC_CAN1_CLK_ENABLE();
          __HAL_RCC_GPIOA_CLK_ENABLE();

          //CAN1 GPIO Configuration
          //PA11     ------> CAN1_RX
          //PA12     ------> CAN1_TX
          GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_12;
          GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
          GPIO_InitStruct.Pull = GPIO_NOPULL;
          GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
          GPIO_InitStruct.Alternate = GPIO_AF9_CAN1;
          HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

          // CAN1 interrupt Init
          HAL_NVIC_SetPriority(CAN1_RX0_IRQn, 0, 0);
          HAL_NVIC_EnableIRQ(CAN1_RX0_IRQn);
        }
      }
      //-------------------------------------------------------------
---------------

      uint8_t TruthCommands(uint8_t* data)
      {
        uint8_t temp_cnt=0;

        if (((TRUTH[8] ^ IN_CAN_OUT_CALL) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_OTBOY) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_ANS_CALL) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_INQ_STATE_AC) == data[8])&&(((TRUTH[4]
^ 1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_SHLF_2) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_RAB_AC) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_INQ_STATE_CA) == data[8])&&(((TRUTH[4]
^ 1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_INQ_STATE_KEYS) ==
data[8])&&(((TRUTH[4] ^ 1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_RESET_KEYS) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_SMENA_KEYS) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_SHLF_1) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
```

```
        if (((TRUTH[8] ^ IN_CAN_SHLF_3) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_MKF_ON) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_MKF_OFF) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_CONTROL_CA) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_RABOTA_CA) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_CONTROL_AC) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_INQ_KN) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;
        if (((TRUTH[8] ^ IN_CAN_CONTROL_BAT) == data[8])&&(((TRUTH[4] ^
1)&0x0F) == (data[4]&0x0F))) return 1;

        if (((TRUTH[8] ^ IN_CAN_APC_ON_OFF) == data[8])&&(((TRUTH[4] ^
2)&0x0F) == (data[4]&0x0F)))
        {
          if (((TRUTH[9] ^ 1) == data[9])||((TRUTH[9] ^ 0) == data[9]))
return 1;
          return 0;
        }
        if (((TRUTH[8] ^ IN_CAN_PROGR_OMSK_MODE) ==
data[8])&&(((TRUTH[4] ^ 2)&0x0F) == (data[4]&0x0F)))
        {
          if (((TRUTH[9] ^ 1) == data[9])||((TRUTH[9] ^ 0) == data[9]))
return 1;
          return 0;
        }
        if (((TRUTH[8] ^ IN_CAN_VOLUME_MKF) == data[8])&&(((TRUTH[4] ^
2)&0x0F) == (data[4]&0x0F)))
        {
          if ((TRUTH[9] ^ 0) == data[9])  return 1;
          if ((TRUTH[9] ^ 1) == data[9])  return 1;
          if ((TRUTH[9] ^ 2) == data[9])  return 1;
          if ((TRUTH[9] ^ 3) == data[9])  return 1;
          if ((TRUTH[9] ^ 4) == data[9])  return 1;
          if ((TRUTH[9] ^ 5) == data[9])  return 1;
          if ((TRUTH[9] ^ 6) == data[9])  return 1;
          if ((TRUTH[9] ^ 7) == data[9])  return 1;
          if ((TRUTH[9] ^ 8) == data[9])  return 1;
          return 0;
        }
        if (((TRUTH[8] ^ IN_CAN_VOLUME_TLF) == data[8])&&(((TRUTH[4] ^
2)&0x0F) == (data[4]&0x0F)))
        {
          if ((TRUTH[9] ^ 0) == data[9])  return 1;
          if ((TRUTH[9] ^ 1) == data[9])  return 1;
          if ((TRUTH[9] ^ 2) == data[9])  return 1;
          if ((TRUTH[9] ^ 3) == data[9])  return 1;
          if ((TRUTH[9] ^ 4) == data[9])  return 1;
          if ((TRUTH[9] ^ 5) == data[9])  return 1;
          if ((TRUTH[9] ^ 6) == data[9])  return 1;
          if ((TRUTH[9] ^ 7) == data[9])  return 1;
          if ((TRUTH[9] ^ 8) == data[9])  return 1;
          return 0;
        }

        if (((TRUTH[8] ^ IN_CAN_DATA_PR_OMSKMODE) ==
data[8])&&(((TRUTH[4] ^ 8)&0x0F) == (data[4]&0x0F)))
        {
```

```
          if ((TRUTH[9]^0xA7)==data[9])
          {
            if
(((TRUTH[10]^1)==data[10])||((TRUTH[10]^2)==data[10])||((TRUTH[10]^3)=
=data[10])||((TRUTH[10]^4)==data[10])||((TRUTH[10]^5)==data[10])||((TR
UTH[10]^6)==
data[10])||((TRUTH[10]^7)==data[10])||((TRUTH[10]^8)==data[10])||((TRU
TH[10]^9)==data[10])||((TRUTH[10]^0xa)==data[10])||((TRUTH[10]^0xb)==d
ata[10])||((TRUTH[10]^0xc)==data[10])||((TRUTH[10]^0xd)==data[10])||((
TRUTH[10]^0xe)==data[10])||((TRUTH[10]^0xf)==data[10])||((TRUTH[10]^0x
10)==data[10])||((TRUTH[10]^0x11)==data[10])||((TRUTH[10]^0x12)==data[
10]))
            {
              return 1;
            }
          }
          return 0;
        }

        if (((TRUTH[8] ^ IN_CAN_ENT_SKN) == data[8])&&(((TRUTH[4] ^
6)&0x0F) == (data[4]&0x0F)))
        {
          if
(((TRUTH[9]^'0')==data[9])||((TRUTH[9]^'1')==data[9])||((TRUTH[9]^'2')
==data[9])||((TRUTH[9]^'3')==data[9])||((TRUTH[9]^'4')==data[9])||((TR
UTH[9]^'5')==
data[9])||((TRUTH[9]^'6')==data[9])||((TRUTH[9]^'7')==data[9])||((TRUT
H[9]^'8')==data[9])||((TRUTH[9]^'9')==data[9]))
          {
            if
(((TRUTH[10]^'0')==data[10])||((TRUTH[10]^'1')==data[10])||((TRUTH[10]
^'2')==data[10])||((TRUTH[10]^'3')==data[10])||((TRUTH[10]^'4')==data[
10])||((TRUTH[10]^'5')==
data[10])||((TRUTH[10]^'6')==data[10])||((TRUTH[10]^'7')==data[10])||(
(TRUTH[10]^'8')==data[10])||((TRUTH[10]^'9')==data[10]))
            {
              if
(((TRUTH[11]^'0')==data[11])||((TRUTH[11]^'1')==data[11])||((TRUTH[11]
^'2')==data[11])||((TRUTH[11]^'3')==data[11])||((TRUTH[11]^'4')==data[
11])||((TRUTH[11]^'5')==
data[11])||((TRUTH[11]^'6')==data[11])||((TRUTH[11]^'7')==data[11])||(
(TRUTH[11]^'8')==data[11])||((TRUTH[11]^'9')==data[11]))
              {
                if
(((TRUTH[12]^'0')==data[12])||((TRUTH[12]^'1')==data[12])||((TRUTH[12]
^'2')==data[12])||((TRUTH[12]^'3')==data[12])||((TRUTH[12]^'4')==data[
12])||((TRUTH[12]^'5')==
data[12])||((TRUTH[12]^'6')==data[12])||((TRUTH[12]^'7')==data[12])||(
(TRUTH[12]^'8')==data[12])||((TRUTH[12]^'9')==data[12]))
                {
                  if
(((TRUTH[13]^'0')==data[13])||((TRUTH[13]^'1')==data[13])||((TRUTH[13]
^'2')==data[13])||((TRUTH[13]^'3')==data[13])||((TRUTH[13]^'4')==data[
13])||((TRUTH[13]^'5')==
data[13])||((TRUTH[13]^'6')==data[13])||((TRUTH[13]^'7')==data[13])||(
(TRUTH[13]^'8')==data[13])||((TRUTH[13]^'9')==data[13]))
                  {
                    return 1;
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```
          return 0;
        }

        if (((TRUTH[8] ^ IN_CAN_DIG_PACK_END) == data[8])&&(((TRUTH[4]
^ 3)&0x0F) == (data[4]&0x0F))) return 1;

        if (((TRUTH[8] ^ IN_CAN_DIG_PACK_1) == data[8])||((TRUTH[8] ^
IN_CAN_DIG_PACK_2) == data[8])||((TRUTH[8] ^ IN_CAN_DIG_PACK_3) ==
data[8])||((TRUTH[8] ^ IN_CAN_DIG_PACK_4) == data[8]))
        {
          temp_cnt = ((TRUTH[4] ^ data[4])&0x0F);

          if ((temp_cnt>=2)&&(temp_cnt<=8))
          {
            for (uint8_t i=0; i < (temp_cnt-1); i++) {
              if
(((TRUTH[i+9]^'0')==data[i+9])||((TRUTH[i+9]^'1')==data[i+9])||((TRUTH
[i+9]^'2')==data[i+9])||((TRUTH[i+9]^'3')==data[i+9])||((TRUTH[i+9]^'4
')==data[i+9])||((TRUTH[i+9]^'5')==
data[i+9])||((TRUTH[i+9]^'6')==data[i+9])||((TRUTH[i+9]^'7')==data[i+9
])||((TRUTH[i+9]^'8')==data[i+9])||((TRUTH[i+9]^'9')==data[i+9]))
              {
                asm(" nop ");
              }
            }
            return 1;
          }
          return 0;
        }

        if (((TRUTH[8] ^ IN_CAN_DIG_PACK_5) == data[8]))
        {
          temp_cnt = ((TRUTH[4] ^ data[4])&0x0F);

          if ((temp_cnt>=2)&&(temp_cnt<=5))
          {
            for (uint8_t i=0; i < (temp_cnt-1); i++) {
              if
(((TRUTH[i+9]^'0')==data[i+9])||((TRUTH[i+9]^'1')==data[i+9])||((TRUTH
[i+9]^'2')==data[i+9])||((TRUTH[i+9]^'3')==data[i+9])||((TRUTH[i+9]^'4
')==data[i+9])||((TRUTH[i+9]^'5')==
data[i+9])||((TRUTH[i+9]^'6')==data[i+9])||((TRUTH[i+9]^'7')==data[i+9
])||((TRUTH[i+9]^'8')==data[i+9])||((TRUTH[i+9]^'9')==data[i+9]))
              {
                asm(" nop ");
              }
              else return 0;
            }
            return 1;
          }
          return 0;
        }

        return 0;

      }
      //-----------------------------------------------------------------
---------------
      void ReceiveCanData(void)
      {
        uint8_t k;
```

```
        uint8_t test_cnt;

        CAN_Receive_Flag = 0;

        //if (!(MCP2517_ReadRegister(C1FIFOSTA2_ADDR) & 1)) return; //
Нет новых сообщений   //test!!!!!!!!!!!!!!!!!!
        while ((MCP2517_ReadRegister(C1FIFOSTA2_ADDR) & 1)) {

          uint32_t ram_addr = MCP25XXFD_RAM_BASE |
MCP2517_ReadRegister(C1FIFOUA2_ADDR);

          uint8_t tmp[16] = {0};
          Truth_flag=1;
          MCP2517_ReadMemory(ram_addr, tmp, 16);
          Truth_flag=0;

          //uint32_t r0 = tmp[0] | (tmp[1] << 8) | (tmp[2] << 16) |
(tmp[3] << 24);   //tmp (0-3) - ID
          uint32_t r1 = tmp[4] | (tmp[5] << 8) | (tmp[6] << 16) |
(tmp[7] << 24);   //tmp (4-7) - DLC and another
          test_cnt = (r1 & 0xFF);   //test_cnt = (r1 & 0xF);

          if (TruthCommands(&tmp[0]))
          {
            for (k=0; k<8; k++) tmp[k+8] = tmp[k+8] ^ TRUTH[k+8];
//unmask tmp

            test_cnt = (test_cnt ^ TRUTH[4]);   //unmask cnt
            test_cnt &= 0xF;

            for (k=0; k<8; k++) REC_FRAME[k]=0xFF;
            for (k=0; k<test_cnt; k++) REC_FRAME[k] = tmp[k+8];

            ICANBufPut(REC_FRAME);
          }

      //    MCP2517_WriteRegister(C1FIFOCON2_ADDR, 1 << 8); //сообщение
прочитано

          uint32_t data_reg = MCP2517_ReadRegister(C1FIFOCON2_ADDR);
          data_reg |= (1 << 8);
          MCP2517_WriteRegister(C1FIFOCON2_ADDR, data_reg);
        }

      }

      /************************ END OF FILE
****************************************/
```

Файл can_spi.c

```
      // Includes ------------------------------------------------
---------------

      #include "can_spi.h"
      #include "spi.h"
      #include "main.h"
      #include "gpio.h"

      #include "stdbool.h"
      #include "stdio.h"
      #include "string.h"
      #include "usart.h"
```

```
#define MCP_INST_RESET                    0x00
#define MCP_INST_WRITE_DATA               0x02
#define MCP_INST_READ_DATA                0x03

void MX_CAN_Init_New(void);
uint32_t MCP2517_ReadRegister(uint16_t addr);
void MCP2517_WriteRegister(uint16_t addr, uint32_t value);
uint8_t RoundingToMul4(uint8_t ch);
void MCP2517_WriteMemory(uint16_t addr, uint8_t* data, uint16_t
len);
void MCP2517_ReadMemory(uint16_t addr, uint8_t* data, uint16_t
len);

/*uint32_t IOCON_data[2] = {
        0x01 << 30,
        0x01 << 28,
};*/

uint32_t C1CON_data[3] = {
        0x04 << 24,
        0x1 << 6,
        0x1 << 5
        //0x01 << 16
        //0x01 << 5, //crcen
     //0x01 << 19 // TXQEN
};

uint32_t C1NBTCFG_data[4] = {
        0x0 << 24, // BRP (Prescaler)
        15 << 16, //TSEG1(Prop + Phase1)
        2  << 8,   //TSEG2(Phase2)
        0x1         //SJW
};

/*uint32_t C1DBTCFG_data[4] = {
        0x00 << 24,      //BRP
        0x15 << 16,     //TSEG1
        0x02 << 8,        //TSEG2
        0x01             //SJW
};
uint32_t C1TDC_data[3] = {
        0x00 << 16,      //TDCV
        0x0F << 8,      //TDCO
        0x02            //TDCMOD
};*/

uint32_t C1INT_data[1] = {
        //0x01 << 27,  //receive FIFO overflow interrupt enable
        //0x01 << 26,  //transmit attempt interrupt enable
        0x01 << 17,  //receive FIFO interrupt enable bit
        //0x01 << 16,  //transmit FIFO interrupt enable bit
};

uint32_t C1FIFOCON1_data[4] = {
//transmit
        0x00 << 29,      // размер полезной нагрузки 8 byte
        0x1F << 24,     // размер очереди 32 mess
        0x03 << 21,     // unlim
        0x01 << 7,      // tx
        //0x01 << 2      // interrupt overflow
};
```

```
uint32_t C1FIFOCON2_data[4] = {                              //receive
        0x00 << 29,  // размер полезной нагрузки
        0x1F << 24,// размер очереди
        0x00 << 7,   // rx
        //0x01 << 2,   // interrupt overflow
        0x01
};

uint32_t C1FLTCON1_data[2] = {
        0x01 << 7,   // Фильтр включен
        0x02       // Привязка фильтра к очереди
};

uint32_t C1FLTOBJ4_data[2] = {
        0x01 << 30,  // extended id filter
        0xB980010 // id  0x417300
};


void MCP2517_Send_Byte(uint8_t code)
{
  while(!LL_SPI_IsActiveFlag_TXE(SPI3))
  {
    asm(" nop ");
  }

  shum_byte = ReadNoiseCan();
  if ((shum_byte & 0x80)) GPIOA->BSRR = GPIO_PIN_2;
  else GPIOA->BRR = GPIO_PIN_2;
  shum_byte = ((shum_byte<<1)&0xff);

  SPI_SendData(SPI3, code);

  while(!LL_SPI_IsActiveFlag_RXNE(SPI3))
  {
    asm(" nop ");
  }

  uint8_t trash = SPI_ReceiveData(SPI3);

}

uint8_t MCP2517_Read_Byte(void)
{
  while(!LL_SPI_IsActiveFlag_TXE(SPI3))
  {
    asm(" nop ");
  }

  shum_byte = ReadNoiseCan();
  shum_byte_r = 0;
  shift_bit = 0x80;
//  shift_cnt = 0;
  if ((shum_byte & 0x80)) GPIOA->BSRR = GPIO_PIN_2;
  else GPIOA->BRR = GPIO_PIN_2;
  shum_byte = ((shum_byte<<1)&0xff);
    asm(" nop ");
    asm(" nop ");
    asm(" nop ");
    asm(" nop ");
    SPI_SendData(SPI3, 0);
```

```
  while(!LL_SPI_IsActiveFlag_RXNE(SPI3))
  {
    asm(" nop ");
  }

  uint8_t byte = SPI_ReceiveData(SPI3);

  if (Truth_flag) {
    TRUTH[(Truth_flag-1)] = shum_byte_r;
    Truth_flag++;
  }
  else byte = (byte ^ shum_byte_r);
  return byte;
}

uint32_t MCP2517_ReadRegister(uint16_t addr)
{
  uint32_t value = 0;

  Can_CS_Low(); //CS low

  MCP2517_Send_Byte((MCP_INST_READ_DATA << 4) | ((addr >> 8) &
0xF)); // READ command
  MCP2517_Send_Byte(addr & 0xFF);

  value |= MCP2517_Read_Byte();
  value |= (uint32_t)MCP2517_Read_Byte() << 8;
  value |= (uint32_t)MCP2517_Read_Byte() << 16;
  value |= (uint32_t)MCP2517_Read_Byte() << 24;

  while(LL_SPI_IsActiveFlag_BSY(SPI3))
  {
    asm(" nop ");
  }

  Can_CS_High(); // CS high

  return value;
}

void MCP2517_WriteMemory(uint16_t addr, uint8_t* data, uint16_t
len)
{
  Can_CS_Low(); //CS low

  MCP2517_Send_Byte(MCP_INST_WRITE_DATA << 4 | ((addr >> 8) &
0xF)); // WRITE command
  MCP2517_Send_Byte(addr & 0xFF);

  for(uint16_t i = 0; i < len; i++)
  {
    MCP2517_Send_Byte(data[i]);
  }

  while(LL_SPI_IsActiveFlag_BSY(SPI3))
  {
    asm(" nop ");
  }

  Can_CS_High(); // CS high
}
void MCP2517_WriteRegister(uint16_t addr, uint32_t value)
```

```
{
  Can_CS_Low(); //CS low

  MCP2517_Send_Byte((MCP_INST_WRITE_DATA << 4) | ((addr >> 8) &
0xF)); // WRITE command
  MCP2517_Send_Byte(addr & 0xFF);

  MCP2517_Send_Byte(value & 0xFF);
  MCP2517_Send_Byte((value >> 8) & 0xFF);
  MCP2517_Send_Byte((value >> 16) & 0xFF);
  MCP2517_Send_Byte((value >> 24) & 0xFF);

  while(LL_SPI_IsActiveFlag_BSY(SPI3))
  {
    asm(" nop ");
  }

  Can_CS_High(); // CS high
}

void MCP2517_WriteRegisterBigTransfer(uint16_t addr ,const
uint32_t* data, uint16_t len)
{
  uint32_t register_value = 0;

  for(int i = 0; i < len; i++)
  {
    register_value |= data[i];
  }

  MCP2517_WriteRegister(addr, register_value);
}

//static inline int MCP2517_WaitFlag(uint32_t register_addr,
uint8_t offset_flag, uint32_t mask, uint32_t target_value, uint32_t
timeout)
//{
//  while((((MCP2517_ReadRegister(register_addr) >> offset_flag)
& mask) != target_value) && --timeout) { }
//  return (timeout == 0) ? 1 : 0;
//}

// Переход в режим Configuration
void MCP2517_EnterConfigMode(void)
{
  uint32_t C1CON_to_write = 0;
  uint32_t C1CON_val = MCP2517_ReadRegister(C1CON_ADDR);

  C1CON_to_write |= ((C1CON_val & 0xF8FFFFFF) | (0x04 << 24));
  MCP2517_WriteRegister(C1CON_ADDR, C1CON_to_write);
}
// Переход в режим Normal 2.0 Mode
void MCP2517_EnterNormalMode(void)
{
  uint32_t C1CON_to_write = 0;
  uint32_t C1CON_val = MCP2517_ReadRegister(C1CON_ADDR);

  C1CON_to_write |= ((C1CON_val & 0xF8FFFFFF) | (0x06 << 24));
  MCP2517_WriteRegister(C1CON_ADDR, C1CON_to_write);
}

// Reset
void MCP2517_Reset(void)
```

```
      {
        Can_CS_Low(); //CS low

        MCP2517_Send_Byte((MCP_INST_RESET << 4) | ((C1CON_ADDR >> 8) &
0xF)); // WRITE command
        MCP2517_Send_Byte(C1CON_ADDR & 0xFF);

        while(LL_SPI_IsActiveFlag_BSY(SPI3))
        {
          asm(" nop ");
        }

        Can_CS_High(); // CS high
      }

      uint8_t MCP2517_Configuration(void)
      {
        MCP2517_WriteRegister(OSC_ADDR, 0x0);

      //  if(MCP2517_WaitFlag(OSC_ADDR,10,0x1,0x1,10000))
      //  {
      //    return 1;
      //  }

        MCP2517_WriteRegister(IOCON_ADDR, 0x0);

        MCP2517_WriteRegister(CRC_ADDR, 0x0);

        MCP2517_WriteRegister(ECCCON_ADDR, 0x0);

        MCP2517_WriteRegisterBigTransfer(C1CON_ADDR, C1CON_data,
sizeof(C1CON_data)/sizeof(C1CON_data[0]));

        MCP2517_WriteRegisterBigTransfer(C1NBTCFG_ADDR, C1NBTCFG_data,
sizeof(C1NBTCFG_data)/sizeof(C1NBTCFG_data[0]));   //ConfigureBitrate

        MCP2517_WriteRegisterBigTransfer(C1FIFOCON1_ADDR,
C1FIFOCON1_data, sizeof(C1FIFOCON1_data)/sizeof(C1FIFOCON1_data[0]));
//передача

        MCP2517_WriteRegisterBigTransfer(C1FIFOCON2_ADDR,
C1FIFOCON2_data, sizeof(C1FIFOCON2_data)/sizeof(C1FIFOCON2_data[0]));
//прием

        MCP2517_WriteRegisterBigTransfer(C1INT_ADDR, C1INT_data,
sizeof(C1INT_data)/sizeof(C1INT_data[0]));

        return 0;
      }

      void MCP2517_ConfigurationFilters(void)
      {
      //  MCP2517_WriteRegister(C1FLTCON1_ADDR, 0x0);

        MCP2517_WriteRegisterBigTransfer(C1FLTOBJ4_ADDR,
C1FLTOBJ4_data, sizeof(C1FLTOBJ4_data)/sizeof(C1FLTOBJ4_data[0]));

        MCP2517_WriteRegister(C1MASK4_ADDR, 0x5FFFFFFF); // MIDE set

        MCP2517_WriteRegisterBigTransfer(C1FLTCON1_ADDR,
C1FLTCON1_data, sizeof(C1FLTCON1_data)/sizeof(C1FLTCON1_data[0]));
      }
```

```
void MCP2517_Init(void)
{
  MCP2517_EnterConfigMode();
  MCP2517_Reset();

  if (MCP2517_Configuration())
  {
    while(1)
    {
      asm(" nop ");
    }
  }

  MCP2517_ConfigurationFilters();
  MCP2517_EnterNormalMode();
}


void MX_CAN_Init_New(void)
{
  Can_CS_High();
  MX_SPI3_Init();  //SPI для работы с MCP2517

  // PB8 - CAN_SCK - прерывание по приему
  __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_8);
  // PC6 - CAN_INT1 - прерывание по приему
  __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_6);

    // EXTI interrupt init
  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

  MCP2517_Init();
}

uint8_t RoundingToMul4(uint8_t ch){
    uint8_t answer = 0;
    while(ch > answer){
        answer += 4;
    }
    return answer;
}

void MCP2517_ReadMemory(uint16_t addr, uint8_t* data, uint16_t
len)
{
  Can_CS_Low(); //CS low

  MCP2517_Send_Byte((MCP_INST_READ_DATA << 4) | ((addr >> 8) &
0xF)); // READ command
  MCP2517_Send_Byte(addr & 0xFF);

  for(uint16_t i = 0; i < len; i++)
  {
    data[i] = MCP2517_Read_Byte();
  }

  while(LL_SPI_IsActiveFlag_BSY(SPI3))
  {
    asm(" nop ");
  }
  Can_CS_High(); // CS high
```

```
}

/************************ END OF FILE
****************************************/
```

Файл usart.c

```
// Includes ------------------------------------------------------
---------------

#include "usart.h"
#include "main.h"
#include "gpio.h"

#include "buffers.h"
#include "font.h"
#include "display_driver.h"
#include "draw_function.h"
#include "events.h"
#include "time_talk.h"
#include "can.h"
#include "menu.h"
#include "widgets.h"
#include "keypad.h"
#include "spi.h"


#define sym_BOF 0xC0
#define sym_EOF 0xC1
#define sym_CE  0x7D
#define MIN_SIZE_PACKET  3
//-----------------------------------------------------------------
---------------
uint8_t TEMP_IN_PACKET[PACKET_LEN];
uint8_t TEMP_OUT_PACKET[PACKET_LEN];
uint8_t OUT_PACKET[PACKET_LEN];
uint8_t L_str=0;
uint16_t L_size=0;
uint8_t L_staff=0;
uint8_t Cnt_mask_rec=0;

uint32_t Addr;
uint16_t Len;


uint8_t IMSPBUF_[IMSPBUFLEN];
uint8_t OMSPBUF_[OMSPBUFLEN];
uint8_t DIGITBUF_[DIGBUFLEN];


UART_HandleTypeDef huart4;

UART_HandleTypeDef huart1;

void MX_UART4_Init(void);
void MX_USART1_UART_Init(void);

uint8_t USART_ReceiveData8(USART_TypeDef *USARTx);
void USART_TransmitData8(USART_TypeDef *USARTx, uint8_t Value);
uint32_t USART_IsActiveFlag_TXE(USART_TypeDef *USARTx);

void SendComm(void);
void TestComm(void);
```

```
      void EX_MAX3221_Init(void);
      void EX_MAX3221_DeInit(void);

      void FormPacketKK(uint8_t code, uint8_t* src, uint8_t* dst,
uint16_t size);

      void AnalysisCommand(void);

      void WriteNoiseCan(uint8_t* data);
      void InquiryCanNoise(void);
      uint8_t ReadNoiseCan(void);
      //-----------------------------------------------------------------
---------------

      // UART4 init function
      void MX_UART4_Init(void)
      {
        huart4.Instance = UART4;
        huart4.Init.BaudRate = 115200;
        huart4.Init.WordLength = UART_WORDLENGTH_8B;
        huart4.Init.StopBits = UART_STOPBITS_1;
        huart4.Init.Parity = UART_PARITY_NONE;
        huart4.Init.Mode = UART_MODE_TX_RX;
        huart4.Init.HwFlowCtl = UART_HWCONTROL_NONE;
        huart4.Init.OverSampling = UART_OVERSAMPLING_16;
        huart4.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
        huart4.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
        if (HAL_UART_Init(&huart4) != HAL_OK)
        {
          Error_Handler();
        }
      }

      //-----------------------------------------------------------------
---------------

      // USART1 init function
      void MX_USART1_UART_Init(void)
      {
        huart1.Instance = USART1;
        huart1.Init.BaudRate = 115200;
        huart1.Init.WordLength = UART_WORDLENGTH_8B;
        huart1.Init.StopBits = UART_STOPBITS_1;
        huart1.Init.Parity = UART_PARITY_NONE;
        huart1.Init.Mode = UART_MODE_TX_RX;
        huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
        huart1.Init.OverSampling = UART_OVERSAMPLING_16;
        huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
        huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
        if (HAL_UART_Init(&huart1) != HAL_OK)
        {
          Error_Handler();
        }
      }

      //-----------------------------------------------------------------
---------------

      void HAL_UART_MspInit(UART_HandleTypeDef* uartHandle)
      {
        GPIO_InitTypeDef GPIO_InitStruct = {0};
        if(uartHandle->Instance==UART4)
```

```
    {
      // UART4 clock enable
      __HAL_RCC_UART4_CLK_ENABLE();
      __HAL_RCC_GPIOA_CLK_ENABLE();

      // UART4 GPIO Configuration
      //PA0     ------> UART4_TX
      //PA1     ------> UART4_RX
      GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
      GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
      GPIO_InitStruct.Pull = GPIO_NOPULL;
      GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
      GPIO_InitStruct.Alternate = GPIO_AF8_UART4;
      HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

      // UART4 interrupt Init
      HAL_NVIC_SetPriority(UART4_IRQn, 1, 0);
      HAL_NVIC_EnableIRQ(UART4_IRQn);
      // Enable the UART Data Register not empty Interrupt
    __HAL_UART_ENABLE_IT(&huart4, UART_IT_RXNE);
    }
    else if(uartHandle->Instance==USART1)
    {
      // USART1 clock enable
      __HAL_RCC_USART1_CLK_ENABLE();
      __HAL_RCC_GPIOA_CLK_ENABLE();
      // USART1 GPIO Configuration
      //PA9     ------> USART1_TX
      //PA10     ------> USART1_RX

      GPIO_InitStruct.Pin = GPIO_PIN_9|GPIO_PIN_10;
      GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
      GPIO_InitStruct.Pull = GPIO_NOPULL;
      GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
      GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
      HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

      // USART1 interrupt Init
      HAL_NVIC_SetPriority(USART1_IRQn, 1, 0);
      HAL_NVIC_EnableIRQ(USART1_IRQn);
      __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);
    }
  }

  //-----------------------------------------------------------------
---------------

  void HAL_UART_MspDeInit(UART_HandleTypeDef* uartHandle)
  {
    if(uartHandle->Instance==UART4)
    {
      // Peripheral clock disable
      __HAL_RCC_UART4_CLK_DISABLE();

      // UART4 GPIO Configuration
      //PA0     ------> UART4_TX
      //PA1     ------> UART4_RX
      HAL_GPIO_DeInit(GPIOA, GPIO_PIN_0|GPIO_PIN_1);

      // UART4 interrupt Deinit
      HAL_NVIC_DisableIRQ(UART4_IRQn);
    }
    else if(uartHandle->Instance==USART1)
```

```
    {
      // Peripheral clock disable
      __HAL_RCC_USART1_CLK_DISABLE();

      // USART1 GPIO Configuration
      //PA9     ------> USART1_TX
      //PA10     ------> USART1_RX
      HAL_GPIO_DeInit(GPIOA, GPIO_PIN_9|GPIO_PIN_10);

      // USART1 interrupt Deinit
      HAL_NVIC_DisableIRQ(USART1_IRQn);
    }
  }

  //-------------------------------------------------------------
---------------

  //
    // @brief  Read Receiver Data register (Receive Data value, 8
bits)
    // @rmtoll DR           DR            LL_USART_ReceiveData8
    // @param  USARTx USART Instance
    // @retval Value between Min_Data=0x00 and Max_Data=0xFF
  //
  uint8_t USART_ReceiveData8(USART_TypeDef *USARTx)
  {
    return (uint8_t)(READ_BIT(USARTx->RDR, USART_DR_DR));
  }

  //-------------------------------------------------------------
---------------

  //
    // @brief  Write in Transmitter Data Register (Transmit Data
value, 8 bits)
    // @rmtoll DR           DR            LL_USART_TransmitData8
    // @param  USARTx USART Instance
    // @param  Value between Min_Data=0x00 and Max_Data=0xFF
    // @retval None
  //
  void USART_TransmitData8(USART_TypeDef *USARTx, uint8_t Value)
  {
    USARTx->TDR = Value;
  }

  //-------------------------------------------------------------
---------------

  //
    // @brief  Check if the USART Transmit Data Register Empty Flag
is set or not
    // @rmtoll SR           TXE           LL_USART_IsActiveFlag_TXE
    // @param  USARTx USART Instance
    // @retval State of bit (1 or 0).
  //
  uint32_t USART_IsActiveFlag_TXE(USART_TypeDef *USARTx)
  {
    return (READ_BIT(USARTx->ISR, USART_ISR_TXE) ==
(USART_ISR_TXE));
  }

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

```
      void SendComm(void)
      {
        uint8_t comm=0;

        if (COMMAND_PORT==1)
        {
          if (USART_IsActiveFlag_TXE(UART4))
          {
            comm = OmspBuf.Ptr[OmspBuf.PtrR++];
            USART_TransmitData8(UART4, comm);
            OmspBuf.Cnt--;
            if(OmspBuf.PtrR>=OMSPBUFLEN) OmspBuf.PtrR=0;
          }
        }
        else
        {
          if (USART_IsActiveFlag_TXE(USART1))
          {
            comm = OmspBuf.Ptr[OmspBuf.PtrR++];
            USART_TransmitData8(USART1, comm);
            OmspBuf.Cnt--;
            if(OmspBuf.PtrR>=OMSPBUFLEN) OmspBuf.PtrR=0;
          }
        }
      }
      //-----------------------------------------------------------------
---------------
      //record param in TEMP_OUT_PACKET
      //and FormPacketKK(OUT_KK_XX_XX, TEMP_OUT_PACKET, OUT_PACKET, u16
size);
      void FormPacketKK(uint8_t code, uint8_t* src, uint8_t* dst,
uint16_t size)
      {
        //  *reform++ = (cnt_value ^ (*mask));
        u16 crc,i;
        u32 new_size,t32;
        u8 cnt_mask=0;

        t32 = (u32)dst;
        crc = crc16new(src,size,(u8)code);
        *dst++ = sym_BOF;

        dst = byte_staffing(dst,(u8)(code ^ MASK[cnt_mask++]));

        for (i = 0; i < size; i++)
        {
          dst = byte_staffing(dst, ((*src & 0xFF) ^ MASK[cnt_mask++]));
          if (cnt_mask==144) cnt_mask=0;
          src++;
        }

        dst = byte_staffing(dst, (((crc>>0)&0xFF) ^ MASK[cnt_mask++])
);
        if (cnt_mask==144) cnt_mask=0;
        dst = byte_staffing(dst, (((crc>>8)&0xFF) ^ MASK[cnt_mask++])
);
        if (cnt_mask==144) cnt_mask=0;
        *dst++ = sym_EOF;

        new_size = (uint32_t)dst;
        new_size -= t32;
        dst = (u8*)t32;
```

```
        for(i=0;i<new_size;i++) CBufPut(&OmspBuf,*dst++);
      }

      //-------------------------------------------------------------
---------------
      void TestComm(void)
      {
        uint16_t rcv_crc,crc;

        TestCommand = CBufGet(&ImspBuf);

        if(TestCommand==sym_BOF)
        {
          clr_com_prebuf();
          L_str = 0xDA;
          return;
        }
        else
        {
          if((TestCommand==sym_EOF)&&(L_str!=0xDA))
          {
            clr_com_prebuf();
            return;
          }
          if((TestCommand!=sym_EOF)&&(L_str==0xDA))
          {
            if(TestCommand==sym_CE)
            {
              L_staff=1;
            }
            else
            {
              if (L_staff)
              {
                L_staff=0;
                TestCommand = TestCommand ^ 0x20;
              }
              TEMP_IN_PACKET[L_size]=(TestCommand ^
MASK[Cnt_mask_rec++]);
              if (Cnt_mask_rec==144) Cnt_mask_rec=0;

              L_size ++;
              if(L_size>=PACKET_LEN)clr_com_prebuf();
            }
            return;
          }
          if ((TestCommand==sym_EOF)&&(L_str==0xDA))
          {
            if(L_size < MIN_SIZE_PACKET)
            {
              clr_com_prebuf();
              return;
            }
            crc = crc16new(TEMP_IN_PACKET,(L_size - 2),0);
            rcv_crc = (TEMP_IN_PACKET[L_size-1]<<8) +
TEMP_IN_PACKET[L_size-2];
            if(rcv_crc != crc)
            {
              //wrong crc
              clr_com_prebuf();
              return;
            }
            AnalysisCommand();
```

```
            clr_com_prebuf();
            return;
          }
        }
      }
      //-----------------------------------------------------------------
---------------

      void EX_MAX3221_DeInit(void)
      {
        EX_EN_GPIO_High_state();
        EX_OFF_GPIO_Low_state();
      }

      void EX_MAX3221_Init(void)
      {
        EX_EN_GPIO_Low_state();       //active
        EX_OFF_GPIO_High_state();     //active
      }


      uint8_t ReadNoiseCan(void)
      {
        uint8_t shum=0;

        uint16_t test = 0;

        if (!CAN_TypePartNoise_Flag)   //если работаем с первой
половиной
        {
          shum = NOISE_CAN_1[CAN_ReadNoise_Ptr++];
          test = (432-CAN_ReadNoise_Ptr);
          if (test <= 50) {
            CAN_ReadNoise_Ptr = 0;
            CAN_StateNoise_Flag_1 = 0;
            CAN_TypePartNoise_Flag = 1;
          }
        }
        else {
          shum = NOISE_CAN_2[CAN_ReadNoise_Ptr++];
          test = (432-CAN_ReadNoise_Ptr);
          if (test <= 50) {
            CAN_ReadNoise_Ptr = 0;
            CAN_StateNoise_Flag_2 = 0;
            CAN_TypePartNoise_Flag = 0;
          }

        }

        return shum;
      }

      void WriteNoiseCan(uint8_t* data)
      {
        uint16_t i;

        CAN_WaitAnswer_Flag = 0;

        if (CAN_TypePartNoise_Flag)   //если работаем со второй
половиной, то заполняем первую
        {
          switch (CAN_StateNoise_Flag_1)
          {
```

```
      case 0:
        for(i=0; i<144; i++)
        {
          NOISE_CAN_1[i] = data[i];
        }
        CAN_StateNoise_Flag_1++;
        break;

      case 1:
        for(i=0; i<144; i++)
        {
          NOISE_CAN_1[i+144] = data[i];
        }
        CAN_StateNoise_Flag_1++;
        break;

      case 2:
        for(i=0; i<144; i++)
        {
          NOISE_CAN_1[i+288] = data[i];
        }
        CAN_StateNoise_Flag_1++;
        break;

      default:
        break;
    } //switch
  }
  else {
    switch (CAN_StateNoise_Flag_2)
    {
      case 0:
        for(i=0; i<144; i++)
        {
          NOISE_CAN_2[i] = data[i];
        }
        CAN_StateNoise_Flag_2++;
        break;

      case 1:
        for(i=0; i<144; i++)
        {
          NOISE_CAN_2[i+144] = data[i];
        }
        CAN_StateNoise_Flag_2++;
        break;

      case 2:
        for(i=0; i<144; i++)
        {
          NOISE_CAN_2[i+288] = data[i];
        }
        CAN_StateNoise_Flag_2++;
        break;

      default:
        break;
    } //switch
  }
}
void InquiryCanNoise(void)
{
```

```
        FormPacketKK(OUT_KK_INQ_NOISE_CAN, TEMP_OUT_PACKET, OUT_PACKET,
0);
        CAN_WaitAnswer_Flag = 1;
      }

      /************************ END OF FILE
****************************************/
```

Файл spi.c

```
      // Includes ------------------------------------------------------
---------------

      #include "spi.h"
      #include "main.h"
      #include "gpio.h"



      SPI_HandleTypeDef hspi1;
      SPI_HandleTypeDef hspi2;
      SPI_HandleTypeDef hspi3;
      void MX_SPI1_Init(void);
      void MX_SPI2_Init(void);
      void MX_SPI3_Init(void);

      uint8_t SPI_ReceiveData(SPI_TypeDef* SPIx);
      void SPI_SendData(SPI_TypeDef* SPIx, uint8_t Data);
      void LL_SPI_Enable(SPI_TypeDef *SPIx);
      uint32_t LL_SPI_IsActiveFlag_BSY(SPI_TypeDef *SPIx);
      uint32_t LL_SPI_IsActiveFlag_TXE(SPI_TypeDef *SPIx);
      uint32_t LL_SPI_IsActiveFlag_RXNE(SPI_TypeDef *SPIx);
      ;


      //-------------------------------------------------------------------
---------------

      uint8_t SPI_ReceiveData(SPI_TypeDef* SPIx)
      {
        uint8_t rez=0;

        // Check the parameters
        assert_param(IS_SPI_ALL_PERIPH(SPIx));

        rez = *(__IO uint8_t *)&SPIx->DR;

        // Return the data in the DR register
        return rez;
      }

      void SPI_SendData(SPI_TypeDef* SPIx, uint8_t Data)
      {
        // Check the parameters
        assert_param(IS_SPI_ALL_PERIPH(SPIx));

        // Write in the DR register the data to be sent
        *(__IO uint8_t *)&SPIx->DR = Data;
      }

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

```
// SPI1 init function
void MX_SPI1_Init(void)
{
  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_16;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 7;
  hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi1.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  LL_SPI_Enable(SPI1);
}

//-------------------------------------------------------------------
---------------

// SPI2 init function
void MX_SPI2_Init(void)
{
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
//  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 7;
  hspi2.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi2.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
  LL_SPI_Enable(SPI2);

  //SPI_FLASH_Erase_All();
}

//-------------------------------------------------------------------
---------------

void HAL_SPI_MspInit(SPI_HandleTypeDef* spiHandle)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(spiHandle->Instance==SPI1)
  {
```

```
    // SPI1 clock enable
    __HAL_RCC_SPI1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    //SPI1 GPIO Configuration
    //PA5     ------> SPI1_SCK
    //PA7     ------> SPI1_MOSI
    GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  }
  else if(spiHandle->Instance==SPI2)
  {
    // SPI2 clock enable
    __HAL_RCC_SPI2_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    //SPI2 GPIO Configuration
    //PB13     ------> SPI2_SCK
    //PB14     ------> SPI2_MISO
    //PB15     ------> SPI2_MOSI
    GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  }
  else if(spiHandle->Instance==SPI3)
  {
    // SPI3 clock enable
    __HAL_RCC_SPI3_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();

    //SPI3 GPIO Configuration
    //PC10     ------> SPI3_SCK
    //PC11     ------> SPI3_MISO
    //PC12     ------> SPI3_MOSI
    GPIO_InitStruct.Pin = GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF6_SPI3;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  }
}
//-------------------------------------------------------------
---------------

void HAL_SPI_MspDeInit(SPI_HandleTypeDef* spiHandle)
{
  if(spiHandle->Instance==SPI1)
  {
    // Peripheral clock disable
    __HAL_RCC_SPI1_CLK_DISABLE();

    //SPI1 GPIO Configuration
    //PA5     ------> SPI1_SCK
    //PA7     ------> SPI1_MOSI
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_5|GPIO_PIN_7);
```

```
    }
    else if(spiHandle->Instance==SPI2)
    {
      // Peripheral clock disable
      __HAL_RCC_SPI2_CLK_DISABLE();

      // SPI2 GPIO Configuration
      //PB13     ------> SPI2_SCK
      //PB14     ------> SPI2_MISO
      //PB15     ------> SPI2_MOSI

      HAL_GPIO_DeInit(GPIOB, GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15);
    }
  }

  //-------------------------------------------------------------------
---------------

  //
    // @brief  Enable SPI peripheral
    // @rmtoll CR1          SPE           LL_SPI_Enable
    // @param  SPIx SPI Instance
    // @retval None
    //
  void LL_SPI_Enable(SPI_TypeDef *SPIx)
  {
    SET_BIT(SPIx->CR1, SPI_CR1_SPE);
  }

  //-------------------------------------------------------------------
---------------

  //
    // @brief  Get busy flag
    // @note   The BSY flag is cleared under any one of the
following conditions:
    // -When the SPI is correctly disabled
    // -When a fault is detected in Master mode (MODF bit set to 1)
    // -In Master mode, when it finishes a data transmission and no
new data is ready to be
    // sent
    // -In Slave mode, when the BSY flag is set to '0' for at least
one SPI clock cycle between
    // each data transfer.
    // @rmtoll SR           BSY           LL_SPI_IsActiveFlag_BSY
    // @param  SPIx SPI Instance
    // @retval State of bit (1 or 0).
    //

  uint32_t LL_SPI_IsActiveFlag_BSY(SPI_TypeDef *SPIx)
  {
    return (READ_BIT(SPIx->SR, SPI_SR_BSY) == (SPI_SR_BSY));
  }

  //
    // @brief  Check if Tx buffer is empty
    // @rmtoll SR           TXE           LL_SPI_IsActiveFlag_TXE
    // @param  SPIx SPI Instance
    // @retval State of bit (1 or 0).
    //
  uint32_t LL_SPI_IsActiveFlag_TXE(SPI_TypeDef *SPIx)
  {
    return (READ_BIT(SPIx->SR, SPI_SR_TXE) == (SPI_SR_TXE));
```

```
}

//
  // @brief  Check if Rx buffer is not empty
  // @rmtoll SR           RXNE          LL_SPI_IsActiveFlag_RXNE
  // @param  SPIx SPI Instance
  // @retval State of bit (1 or 0).
  //
uint32_t LL_SPI_IsActiveFlag_RXNE(SPI_TypeDef *SPIx)
{
  return (READ_BIT(SPIx->SR, SPI_SR_RXNE) == (SPI_SR_RXNE));
}


//---------------------------------------------------------------
---------------

// SPI3 init function
void MX_SPI3_Init(void) {

  hspi3.Instance = SPI3;
  hspi3.Init.Mode = SPI_MODE_MASTER;
  hspi3.Init.Direction = SPI_DIRECTION_2LINES;
  hspi3.Init.DataSize = SPI_DATASIZE_8BIT;

  hspi3.Init.CLKPolarity = SPI_POLARITY_HIGH;
  hspi3.Init.CLKPhase = SPI_PHASE_2EDGE;

  hspi3.Init.NSS = SPI_NSS_SOFT;
  hspi3.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_128;
  hspi3.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi3.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi3.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi3.Init.CRCPolynomial = 7;
  hspi3.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi3.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
  if (HAL_SPI_Init(&hspi3) != HAL_OK)
  {
    Error_Handler();
  }
  LL_SPI_Enable(SPI3);

}

/************************ END OF FILE
****************************************/
```

Файл stm32l4xx_it.c

```
// Includes ------------------------------------------------------
---------------
#include "stm32l4xx_it.h"
#include "main.h"
#include "keypad.h"

#include "usart.h"
#include "buffers.h"
#include "gpio.h"
#include "can.h"

uint8_t ButFreq=0;
//uint8_t REC_FRAME[18];
```

```
      //-------------------------------------------------------------
---------------
      //           Cortex-M4 Processor Interruption and Exception
Handlers
      //-------------------------------------------------------------
---------------

      void NMI_Handler(void);
      void HardFault_Handler(void);
      void MemManage_Handler(void);
      void BusFault_Handler(void);
      void UsageFault_Handler(void);
      void SVC_Handler(void);
      void DebugMon_Handler(void);
      void PendSV_Handler(void);
      void SysTick_Handler(void);
      void USART1_IRQHandler(void);
      void UART4_IRQHandler(void);
      void UART5_IRQHandler(void);
      void CAN1_RX0_IRQHandler(void);
      void CAN1_SCE_IRQHandler(void);
      void EXTI9_5_IRQHandler(void);

      //  @brief This function handles Non maskable interrupt.

      void NMI_Handler(void)
      {
      //    asm(" nop ");

        while (1)
        {
          asm(" nop ");
        }
      }

      //-------------------------------------------------------------
---------------


      // @brief This function handles Hard fault interrupt.

      void HardFault_Handler(void)
      {
        while (1)
        {
          asm(" nop ");
        }
      }

      //-------------------------------------------------------------
---------------

      // @brief This function handles Memory management fault.

      void MemManage_Handler(void)
      {
        while (1)
        {
          asm(" nop ");
        }
      }
```

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

      //  @brief This function handles Prefetch fault, memory access
fault.

      void BusFault_Handler(void)
      {
        while (1)
        {
          asm(" nop ");
        }
      }

      //-------------------------------------------------------------
---------------

      // @brief This function handles Undefined instruction or illegal
state.

      void UsageFault_Handler(void)
      {
        while (1)
        {
          asm(" nop ");
        }
      }

      //-------------------------------------------------------------
---------------

      // @brief This function handles System service call via SWI
instruction.

      void SVC_Handler(void)
      {
        asm(" nop ");
      }

      //-------------------------------------------------------------
---------------

      // @brief This function handles Debug monitor.

      void DebugMon_Handler(void)
      {
        asm(" nop ");
      }

      //-------------------------------------------------------------
---------------

      // @brief This function handles Pendable request for system
service.

      void PendSV_Handler(void)
      {
        asm(" nop ");
      }
```

```
      // @brief This function handles System tick timer.

      void SysTick_Handler(void)
      {
        HAL_IncTick();

        if (Sys_TimingDelay) Sys_TimingDelay--;

        if (ButTimer) ButTimer--;
        else if(!ButNopress)
        {
          ButFreq++;
          if(ButFreq == 16)
          {
            ButFreq = 0;
            ButRunNull++;
            if(ButRunNull == 4) ButRunNull = 1;

            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2,
GPIO_PIN_SET);
            if(ButRunNull == 1) HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0,
GPIO_PIN_RESET);
            else if(ButRunNull == 2) HAL_GPIO_WritePin(GPIOB,
GPIO_PIN_1, GPIO_PIN_RESET);
            else if(ButRunNull == 3) HAL_GPIO_WritePin(GPIOB,
GPIO_PIN_2, GPIO_PIN_RESET);
          }
        }

        if (NsdUprTimer) NsdUprTimer--;
        if (CursTimer) CursTimer--;
        if (EventTimer) EventTimer--;
        if (StatusTimer) StatusTimer--;
        if (TimeTimer) TimeTimer--;
        if (CE_Timer) CE_Timer--;
        if (KoshPullTimer) KoshPullTimer--;
        if (DisplayTimer) DisplayTimer--;
        if (MenuTimer) MenuTimer--;
        if (PressingTimer) PressingTimer--;
        if (SvDiodTimer) SvDiodTimer--;

        if (Timer1000 == 1000)
        {
          Timer1000=0;
          if (EnableTimeTalk) TIME_TALK[7]++;

      /*    if (!TestParam) TestParam=20;
          printf("TestParam:%d\r\n",TestParam);
          TestParam--;*/
        }
        else Timer1000++;
      }

      /******************************************************************
**************/
      /* STM32L4xx Peripheral Interrupt Handlers
*/
      /* Add here the Interrupt Handlers for the used peripherals.
*/
      /* For the available peripheral interrupt handler names,
*/
      /* please refer to the startup file (startup_stm32l4xx.s).
*/
```

```
/*****************************************************************
**************/

//
  // @brief This function handles USART1 global interrupt.
//
void USART1_IRQHandler(void)
{
  uint32_t status = USART1->ISR;
  uint8_t DataUart=USART1->RDR;

  if (status & USART_ISR_ORE)
  {
    asm(" nop ");
    asm(" nop ");
  }
  if (!COMMAND_PORT)
  {
    if (status & USART_ISR_RXNE)
    {
      CBufPut(&ImspBuf,DataUart);
    }
  }
}

//---------------------------------------------------------------
---------------

//
  // @brief This function handles UART4 global interrupt.
//
void UART4_IRQHandler(void)
{
  uint32_t status = UART4->ISR;
  uint8_t DataUart=UART4->RDR;

  if (status & USART_ISR_ORE)
  {
    asm(" nop ");
    asm(" nop ");
  }
  if (COMMAND_PORT==1)
  {
    if (status & USART_ISR_RXNE)
    {
      CBufPut(&ImspBuf,DataUart);
    }
  }
}


void EXTI9_5_IRQHandler(void)
{
  if (EXTI->PR1 & EXTI_PR1_PIF8)
  {
    EXTI->PR1 |= EXTI_PR1_PIF8;

 //   shift_cnt ++;
    if ((GPIOA->IDR & GPIO_PIN_2)) shum_byte_r |= shift_bit;
    else shum_byte_r &= (~shift_bit);
    shift_bit = ((shift_bit>>1) &0x7f);
    if ((shum_byte & 0x80)) GPIOA->BSRR = GPIO_PIN_2;
```

```
          else GPIOA->BRR = GPIO_PIN_2;
          shum_byte = ((shum_byte<<1)&0xff);
        }

        if (EXTI->PR1 & EXTI_PR1_PIF6)
        {
          CAN_Receive_Flag=1;
          EXTI->PR1 |= EXTI_PR1_PIF6;
        }


      //  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_6);
      //  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_8);
      }


      /************************ END OF FILE
**************************************/
```

Файл buffers.c

```
      // Includes ----------------------------------------------------
---------------

      #include "buffers.h"

      // -------------------------------------------------------------
---------------

      CBuf ImspBuf;
      CBuf OmspBuf;
      CBuf DigitBuf;


      // прототипы функций для работы с буферами
      void CBufInit(struct CBuf* Buf, uint8_t* Begin, uint16_t Len);
// функция для создания и инициализации циклического буфера
      void CBufPut(struct CBuf *Buf, uint8_t data);
// функция для записи элемента в буфер
      uint8_t CBufGet(struct CBuf *Buf);
// функция для чтения элемента из буфера

      //------ Инициализация буфера ----------------------------------
---------------
      void CBufInit(struct CBuf* Buf, uint8_t* Begin, uint16_t Len)
      {
        (*Buf).Ptr=Begin;
        (*Buf).PtrR=0;
        (*Buf).PtrW=0;
        (*Buf).Cnt=0;
        (*Buf).Len=Len;

        for(uint16_t i=0;i<(*Buf).Len;i++){(*Buf).Ptr[i]=0;}
      }
      //------------------- Запись в буфер ------------------

      void CBufPut(struct CBuf *Buf, uint8_t  data)
      {
        (*Buf).Ptr[ (*Buf).PtrW ]=data;
        (*Buf).PtrW++;
        (*Buf).Cnt++;
        if( (*Buf).PtrW==(*Buf).Len ) //если указатель записи дошёл до
последнего элемента буфера, указатель записи сбрасывается в ноль.
```

```
  (*Buf).PtrW=0;
}
//------------------- Чтение из буфера ----------------
uint8_t CBufGet(struct CBuf *Buf)
{
  uint8_t data;
  data=(*Buf).Ptr[ (*Buf).PtrR ];
  (*Buf).PtrR++;
  (*Buf).Cnt--;
  if( (*Buf).PtrR==(*Buf).Len ) // если указатель чтения дошёл до
последнего элемента буфера, указатель чтения сбрасывается в ноль.
  (*Buf).PtrR=0;
  return data;
}
//-----------------------------------------------------------------
---------------
```

Файл gpio.c

```
// Includes ------------------------------------------------------
---------------
#include "gpio.h"
#include "main.h"
//#include "usart.h"

//-----------------------------------------------------------------
---------------
void MX_GPIO_Init(void);

uint32_t LL_GPIO_IsInputPinSet(GPIO_TypeDef *GPIOx, uint32_t
PinMask);
void GPIO_WriteOutputPort(GPIO_TypeDef *GPIOx, uint32_t
PortValue);
uint32_t GPIO_ReadOutputPort(GPIO_TypeDef *GPIOx);

void DC_GPIO_Command(void);
void DC_GPIO_Data(void);
void D_RS_GPIO_Low_state(void);
void D_RS_GPIO_High_state(void);

void D_CS_GPIO_Low_state(void);
void D_CS_GPIO_High_state(void);

void D_ON_GPIO_Low_state(void);
void D_ON_GPIO_High_state(void);


void Red_On(void);
void Red_Off(void);

void Green_On(void);
void Green_Off(void);

void sFlash_CS_High(void);
void sFlash_CS_Low(void);

void Can_CS_High(void);
void Can_CS_Low(void);

void Can_Stby_Off(void);
void Can_Stby_On(void);
```

```
      void EX_OFF_GPIO_Low_state(void);
      void EX_OFF_GPIO_High_state(void);

      void EX_EN_GPIO_Low_state(void);
      void EX_EN_GPIO_High_state(void);



      //-----------------------------------------------------------------
---------------
      // Configure GPIO
      //-----------------------------------------------------------------
---------------

      // Configure pins as
      //        * Analog
      //        * Input
      //        * Output
      //        * EVENT_OUT
      //        * EXTI

      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();
        __HAL_RCC_GPIOD_CLK_ENABLE();

        // Configure GPIO pin Output Level
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_8,
GPIO_PIN_RESET);

        // Configure GPIO pin Output Level
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4|GPIO_PIN_6, GPIO_PIN_SET);

        // Configure GPIO pin Output Level
        HAL_GPIO_WritePin(GPIOB,
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_12
                                |GPIO_PIN_6, GPIO_PIN_SET);

        // Configure GPIO pin Output Level
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_SET);

        // Configure GPIO pin Output Level
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9,
GPIO_PIN_RESET);

        // Configure GPIO pin Output Level
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);

        // Configure GPIO pins : PC13 PC0 PC1
        GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_0|GPIO_PIN_1;
        GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

        // Configure GPIO pins : PA6 PA8
        GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_8;
        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 : PA2 PA3 PA4
        GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

        // Configure GPIO pins : PB0 PB1 PB2 PB12
        GPIO_InitStruct.Pin =
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_12;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
        HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

        // Configure GPIO pins : PC7 PC8
        GPIO_InitStruct.Pin = GPIO_PIN_7|GPIO_PIN_8;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
        HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

        // Configure GPIO pin : PC9
        GPIO_InitStruct.Pin = GPIO_PIN_9;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
        HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

        // Configure GPIO pins : PB5 PB6
        GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
        HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

        // Configure GPIO pin : PB7
        GPIO_InitStruct.Pin = GPIO_PIN_7;
        GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

        // Configure GPIO pins : PD2
        GPIO_InitStruct.Pin = GPIO_PIN_2;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
        HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

        // Configure GPIO pins : PC6
        GPIO_InitStruct.Pin = GPIO_PIN_6;
        GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

        // Configure GPIO pins : PB8
        GPIO_InitStruct.Pin = GPIO_PIN_8;
        GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
        GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
```

```
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  }

  //-------------------------------------------------------------
-------------

  //
    // @brief  Return if input data level for several pins of
dedicated port is high or low.
    // @rmtoll IDR          IDy          LL_GPIO_IsInputPinSet
    // @param  GPIOx GPIO Port
    // @param  PinMask This parameter can be a combination of the
following values:
    //         @arg @ref LL_GPIO_PIN_0
    //         @arg @ref LL_GPIO_PIN_1
    //         @arg @ref LL_GPIO_PIN_2
    //         @arg @ref LL_GPIO_PIN_3
    //         @arg @ref LL_GPIO_PIN_4
    //         @arg @ref LL_GPIO_PIN_5
    //         @arg @ref LL_GPIO_PIN_6
    //         @arg @ref LL_GPIO_PIN_7
    //         @arg @ref LL_GPIO_PIN_8
    //         @arg @ref LL_GPIO_PIN_9
    //         @arg @ref LL_GPIO_PIN_10
    //         @arg @ref LL_GPIO_PIN_11
    //         @arg @ref LL_GPIO_PIN_12
    //         @arg @ref LL_GPIO_PIN_13
    //         @arg @ref LL_GPIO_PIN_14
    //         @arg @ref LL_GPIO_PIN_15
    //         @arg @ref LL_GPIO_PIN_ALL
    // @retval State of bit (1 or 0).
  //
  uint32_t LL_GPIO_IsInputPinSet(GPIO_TypeDef *GPIOx, uint32_t
PinMask)
  {
    return (READ_BIT(GPIOx->IDR, PinMask) == (PinMask));
  }

  //-------------------------------------------------------------
-------------

  //
    // @brief  Write output data register for the port.
    // @param  GPIOx GPIO Port
    // @param  PortValue Level value for each pin of the port
    // @retval None
  //
  void GPIO_WriteOutputPort(GPIO_TypeDef *GPIOx, uint32_t
PortValue)
  {
    GPIOx->ODR = PortValue;
  }

  //-------------------------------------------------------------
-------------

  //
    // @brief  Return full output data register value for a
dedicated port.
    // @param  GPIOx GPIO Port
    // @retval Output data register value of port
```

```
      //
      uint32_t GPIO_ReadOutputPort(GPIO_TypeDef *GPIOx)
      {
        return (GPIOx->ODR);
      }

      //-------------------------------------------------------------
-------------

      void DC_GPIO_Command(void)
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
      }

      void DC_GPIO_Data(void)
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);
      }

      //-------------------------------------------------------------
-------------

      void D_RS_GPIO_Low_state(void)
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
      }

      void D_RS_GPIO_High_state(void)
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
      }


      //-------------------------------------------------------------
-------------

      void D_CS_GPIO_Low_state(void)
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
      }

      void D_CS_GPIO_High_state(void)
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
      }
      //-------------------------------------------------------------
-------------

      void D_ON_GPIO_Low_state(void)
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
      }

      void D_ON_GPIO_High_state(void)
      {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
      }

      //-------------------------------------------------------------
-------------
      void Red_Off(void)
```

```
{
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_RESET);
}

void Red_On(void)
{
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_SET);
}


//-------------------------------------------------------------------
-------------

void Green_Off(void)
{
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);
}

void Green_On(void)
{
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);
}

//-------------------------------------------------------------------
-------------

void sFlash_CS_High(void)
{
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET);
}

//-------------------------------------------------------------------
---------------

void sFlash_CS_Low(void)
{
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
}

//-------------------------------------------------------------------
-------------

void Can_Stby_On(void)
{
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, GPIO_PIN_RESET);
}

void Can_Stby_Off(void)
{
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, GPIO_PIN_SET);
}


//-------------------------------------------------------------------
-------------

void EX_OFF_GPIO_Low_state(void)
{
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
}

void EX_OFF_GPIO_High_state(void)
{
```

```
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
      }

      //-------------------------------------------------------------
-------------

      void EX_EN_GPIO_Low_state(void)
      {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
      }

      void EX_EN_GPIO_High_state(void)
      {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
      }

      //-------------------------------------------------------------
-------------

      void Can_CS_High(void)
      {
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_SET);
      }

      //-------------------------------------------------------------
---------------

      void Can_CS_Low(void)
      {
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_RESET);
      }


      /***************************** END OF FILE
**********************************/
```

Приложение Г

(обязательное)

Преобразователь логических сигналов между шинами SPI и CAN

Презентация

Выпускная квалификационная работа бакалавра

ПРЕОБРАЗОВАТЕЛЬ ЛОГИЧЕСКИХ СИГНАЛОВ МЕЖДУ ШИНАМИ SPI И CAN

Автор работы: Студент гр. 22ВВП1 Перфилов А.В.

Научный руководитель: доцент, к.т.н. Егоров В.Ю.

Постановка задачи

Проблема:

Возможна активация закладки в контроллере иностранного производителя через команды передаваемые по шине CAN.

Решение:

Разработать узел контроллера CAN между контроллером и АРМ, выполняющий функции преобразователя между шинами интерфейсов SPI и CAN, а также маскирования шины SPI.

Общая структура аппаратуры

Аппаратное решение

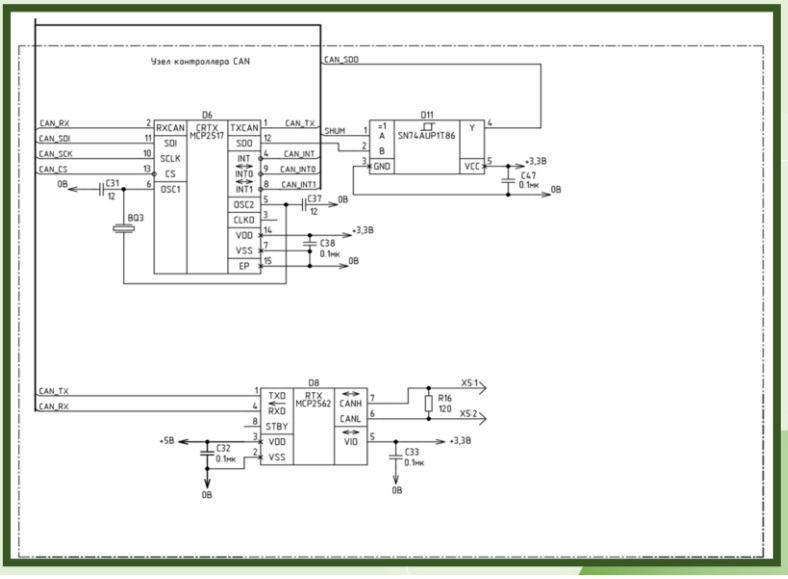
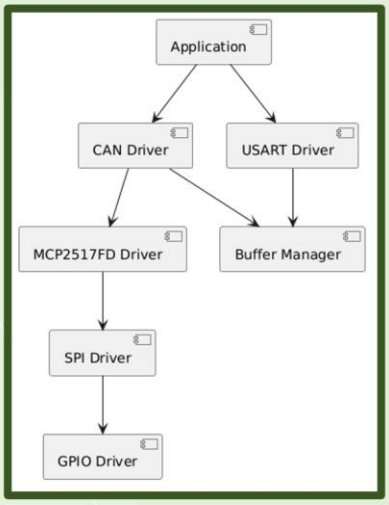
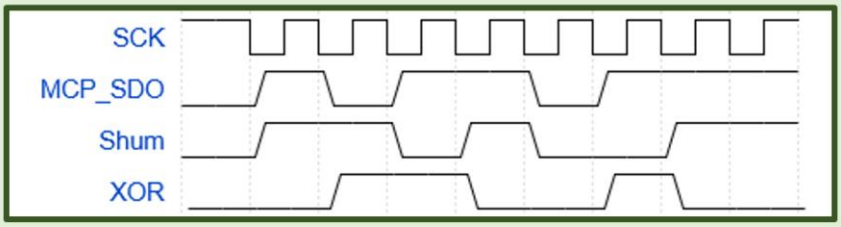
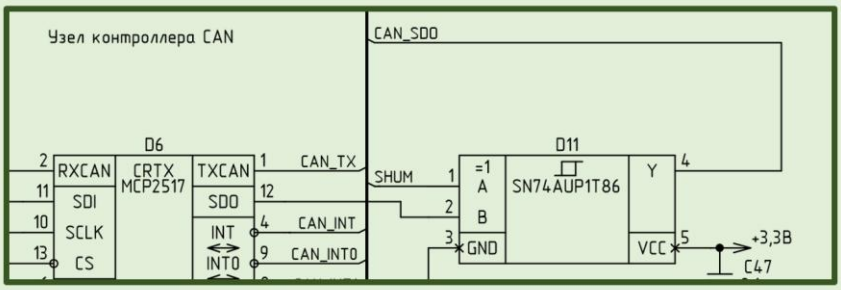


Диаграмма компонентов

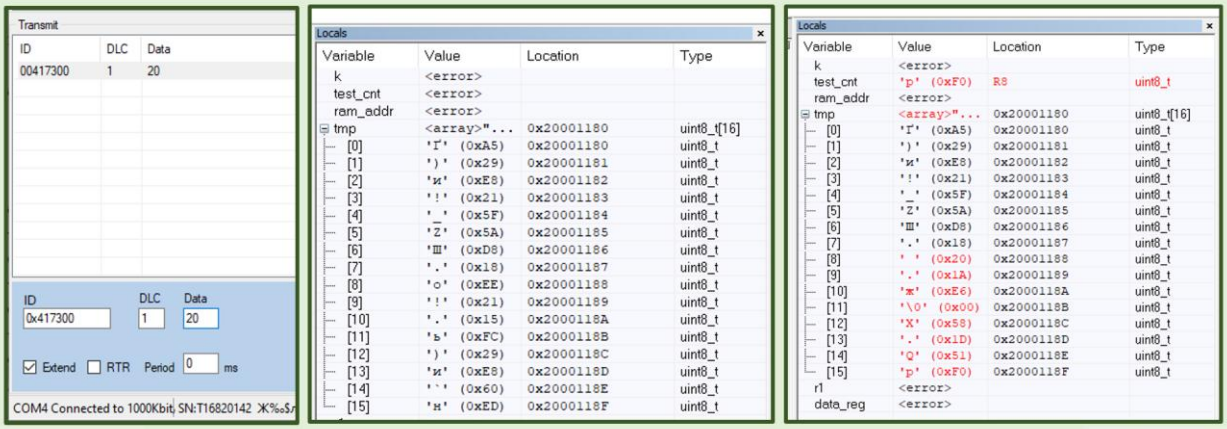


Временная диаграмма

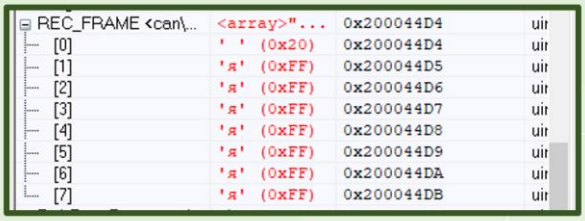


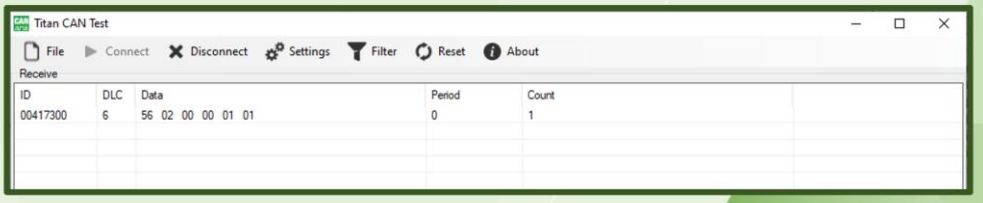


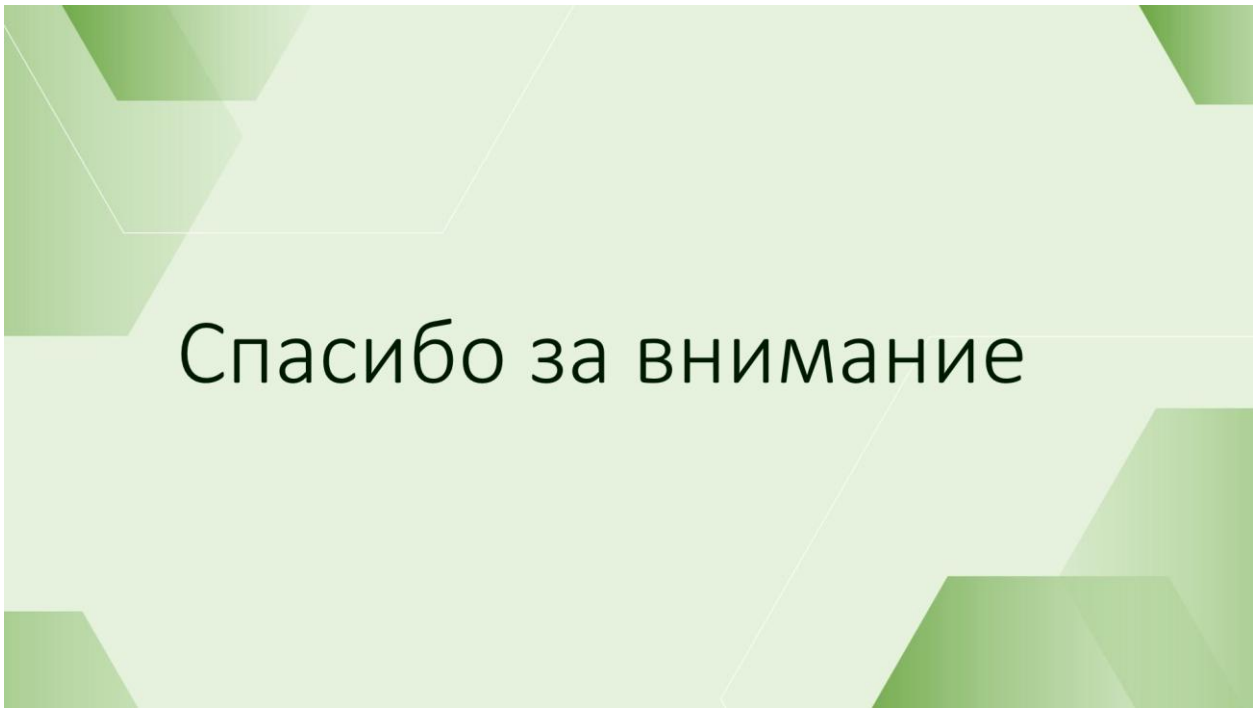
Тестирование



Тестирование






Спасибо за внимание