Конфигурационный план взаимодействия компонентов. Протокол взаимодействия и сигнализации компонентов системы. Архитектура встраиваемого программного обеспечения.

Архитектура встраиваемого программного обеспечения для плат СЦ, СТ, СЛ.

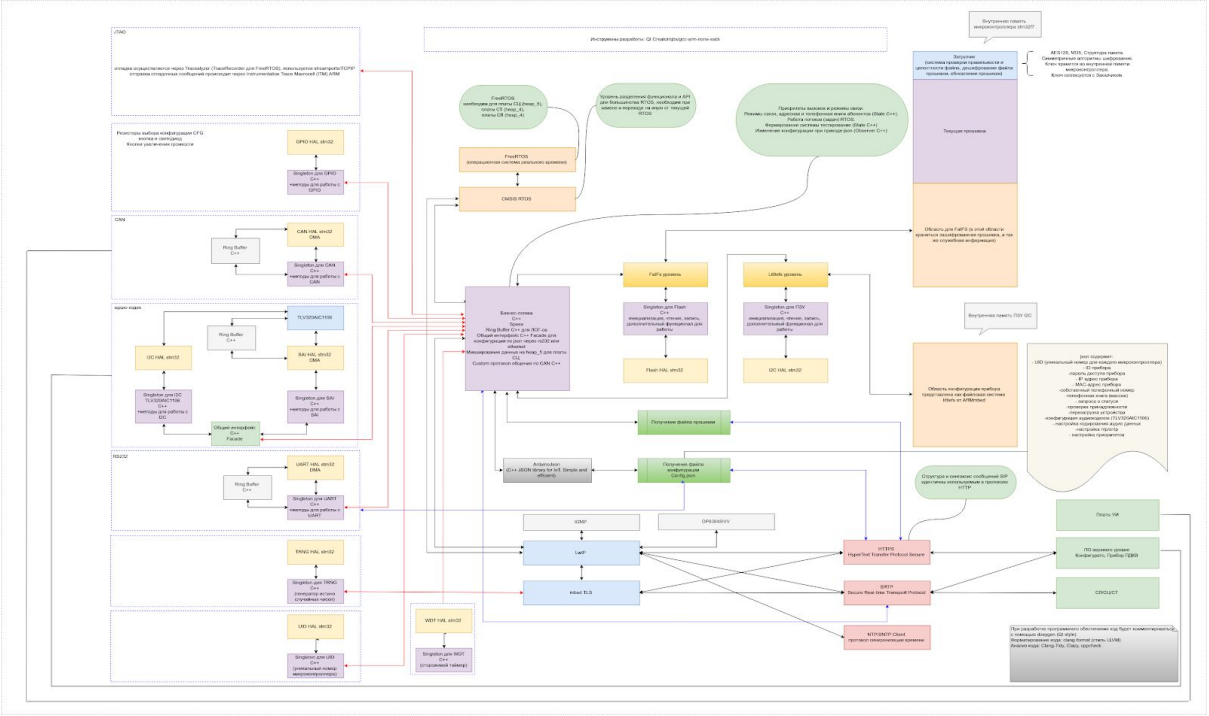

Рисунок 1. Архитектура ВПО (встраиваемого программного обеспечения) для плат СЦ, СТ, СЛ.

1. Процедура обновления ВПО:
    a. При подаче питание загружается bootloader (загрузчик), bootloader монтирует часть область внутренней памяти микроконтроллера, которая представлена файловой системой FatFs.
    b. Загрузчик находит файл прошивки, если файла прошивки нету, загрузчик передает управление области “Текущая прошивка”, если прошивка есть, происходит проверка версионности прошивки, если прошивки одинаковые, загрузчик передает управление текущей прошивки, если прошивки разные, загрузчик проверяет структуру прошивки, далее загрузчик рассчитывает md5 от прошивки, а так же расшифровывает прошивку (алгоритм шифрования AES128), если все операции проведены корректно, тогда прошивка обновляется (рисунок 2). После происходит проверка обновления ВПО, поле версии ВПО записывается сам последним, чтобы гарантировать целостность обновленных данных.

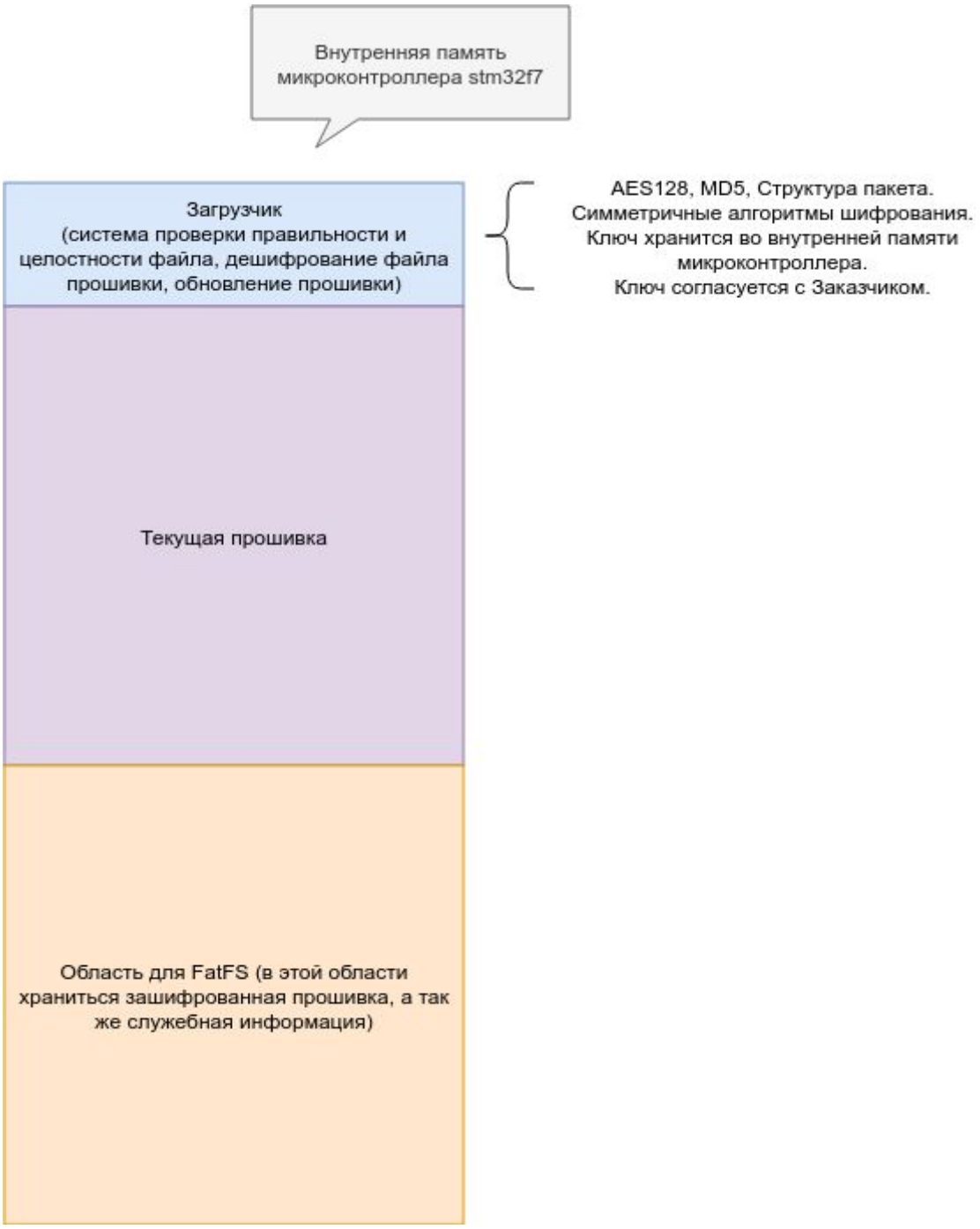

Рисунок 2. Внутреннее представление памяти микроконтроллера серии stm32f7.

Описание	Размер
Версия новой прошивки Версия.Порядковый номер обновления	4B
Размер новой прошивки	4B
MD5	16B

Время и дата создание прошивки	30B
Резервная область (возможно добавление новых полей в будущем)	202B
Зашифрованная новая прошивка (AES128)	Размер новой прошивки

Структура файла прошивки.

2. Процедура загрузки новой прошивки.

a. При старте текущей прошивки происходит монтирование область с FatFs. Новая прошивка загружается по HTTPS с помощью метода GET и файлом помещается в область FatFs.

b. Обновление прошивки происходит только после перезагрузки микроконтроллера.

3. Уровень представления памяти микроконтроллера и ПЗУ (внешняя память).

a. Внутренняя память микроконтроллера разбита на 3 части: Загрузчик, Текущая прошивка, FatFs.

b. Область ПЗУ (внешняя память), представлена как файловая система littlefs.

c. Уровень взаимодействия через файловые системы был выбран как наиболее унифицированный и платформонезависимый, уровень представления позволяет с более высокой легкостью мигрировать между микросхемами памяти и микроконтроллерами и дает более высокий уровень взаимодействия и понимания информации.

d. Нижний уровень взаимодействия представлен через паттерн проектирования Singleton (класс сам контролирует процесс создания единственного экземпляра, паттерн легко адаптировать для создания нужного числа экземпляров, возможность создания объектов классов, производных от Singleton). Применяется для разделения уровня HAL stm32 (Flash и I2C) и уровня взаимодействия с файловыми системами (рисунок 3).

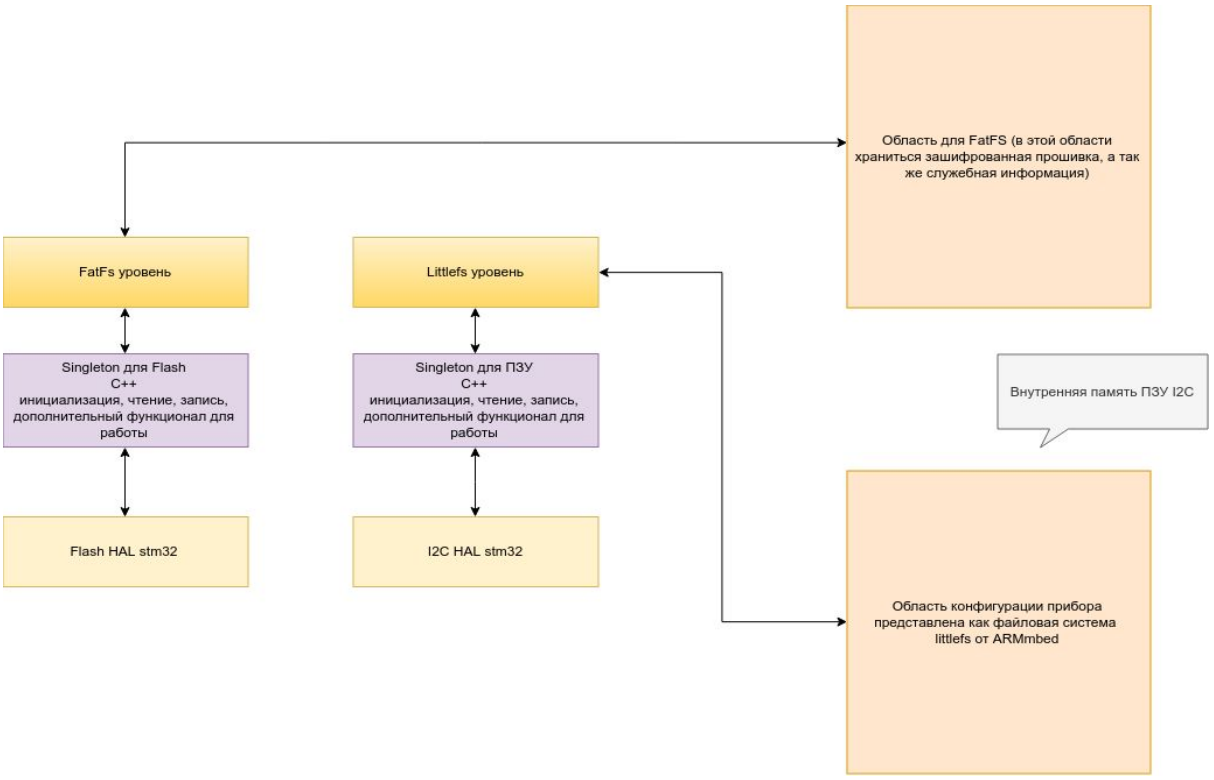

Рисунок 3. Представление взаимодействия с файловыми системами (FatFs/LittleFs).

4. Уровень взаимодействия по CAN:
   a. Нижний уровень взаимодействия представлен через паттерн проектирования Singleton. Применяется для разделения уровня HAL CAN stm32 и уровня взаимодействия с бизнес логикой.
   b. Отправка и получение данных по CAN происходят благодаря механизму DMA. Полученные данные через DMA заносятся в кольцевой буфер (кольцевой буфер эффективных способов организовать FIFO без использования динамической памяти). CAN интерфейс используется для обмена данными с платами УИ. Для обмена данными используется custom протокол.

Номер байта	Тип данных	Содержание	Значение	Описание
0	uint8_t magic	Маркер старта пакета	0xFA	Байт начала транзакции при передачи сообщения, данный байт свидетельствует о правильной структуре передаваемого пакета.
1-2	uint16_t len	Длина полезных	0-1024	Размер данных (полезная

		данных		нагрузка).
3 - 4	uin16_t seq	Номер пакета		Содержит уникальный номер передаваемого сообщения, каждое сообщение инкриминируется.
5	uint8_t type	Тип пакета		Тип передаваемого пакета, 0 — ведомые устройства присылают информацию о своем состоянии, 1 - ведущее устройство посылает информацию.
6	uint8_t id	ID платы		Идентификатор платы, данное значение записывается после установки резисторов конфигурации.
7 — (7 + n) n → [0 - 1024]	uint8_t payload[0 -1024]	Полезные данные		Полезные данные сообщения, информация о нажатие кнопки, установка режима мигания светодиодов.
(n + 8) — (n + 9)	uint16_t crc	Чек сумма		crc16 ccitt - алгоритм расчета чек суммы.

Протокол обмена информацией по CAN.

Протокол обмена информацией по CAN может быть незначительно изменен, могут быть изменения которые касаются поля type, а так же появится специализированная структура payload, которая детализирует размещение структурных данных в данных payload.

5. Уровень взаимодействия с аудио кодеком TLV320AIC3254:
   a. Уровень передачи аудиопотока TLV320AIC3254. Нижний уровень взаимодействия представлен через паттерн проектирования Singleton. Применяется для разделения уровня HAL SAI stm32 и уровня взаимодействия с бизнес логикой.
   b. Отправка и получение данных по SAI происходят благодаря механизму DMA. Полученные данные через DMA заносятся в кольцевой буфер.

c. Уровень конфигурирования аудио кодека TLV320AIC3254. Нижний уровень взаимодействия представлен через паттерн проектирования Singleton. Применяется для разделения уровня HAL I2C stm32 и уровня взаимодействия с бизнес логикой.
d. Создание общего (обобщенного) интерфейса для аудиокодека через паттерн Facade (определяет унифицированный высокоуровневый интерфейс к подсистеме аудиокодека, что упрощает использование Singleton TLV320AIC3254 I2C и Singleton TLV320AIC3254 SAI). Facade "обертывает" эту подсистему более простым интерфейсом (рисунок 4).

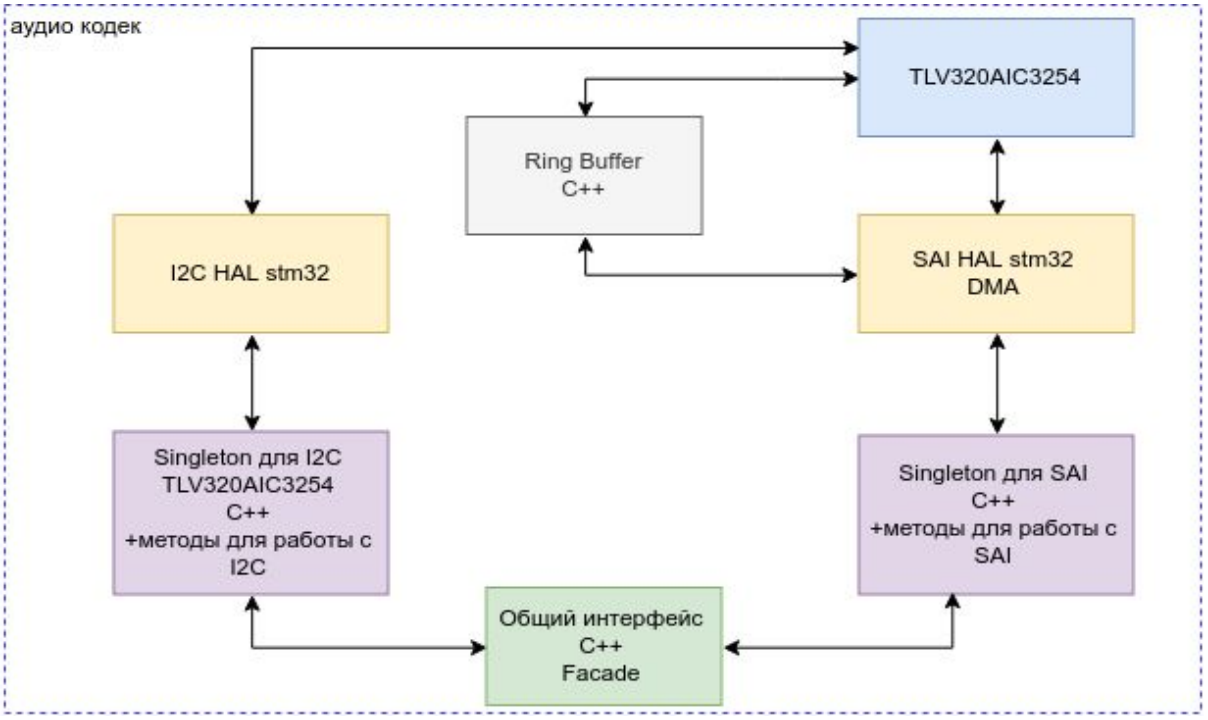

Рисунок 4. Внутреннее представление взаимодействия с аудио кодеком TLV320AIC3254.

6. Уровень взаимодействия с RS232:
a. Нижний уровень взаимодействия представлен через паттерн проектирования Singleton. Применяется для разделения уровня HAL UART stm32 и уровня взаимодействия с бизнес логикой.
b. Отправка и получение данных по UART происходят благодаря механизму DMA. Полученные данные через DMA заносятся в кольцевой буфер (рисунок 5).
c. Через данный интерфейс происходит конфигурирование системы целиком, обмен данными происходит через JSON, со стороны верхнего уровня передается JSON сообщение, данные сообщение является файлом конфигурации Config.json, и сохраняется на внешней ПЗУ, которая представлена как LittleFS. Парсинг сообщений происходит благодаря библиотеке ArduinoJSON. Singleton

включает методы потокового обмена сообщениями, является аналогом операторов cout/cin C++.

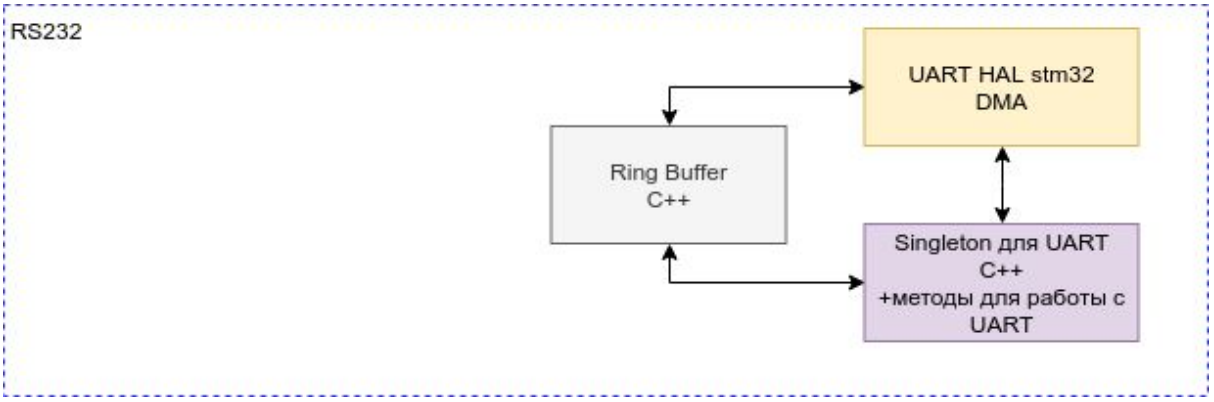

Рисунок 5. Внутреннее представление взаимодействия по RS232.

7. Уровень взаимодействия для работы с генератором случайных чисел:
   a. Нижний уровень взаимодействия представлен через паттерн проектирования Singleton. Применяется для разделения уровня HAL TRNG STM32 и уровня взаимодействия с бизнес логикой (рисунок 6).

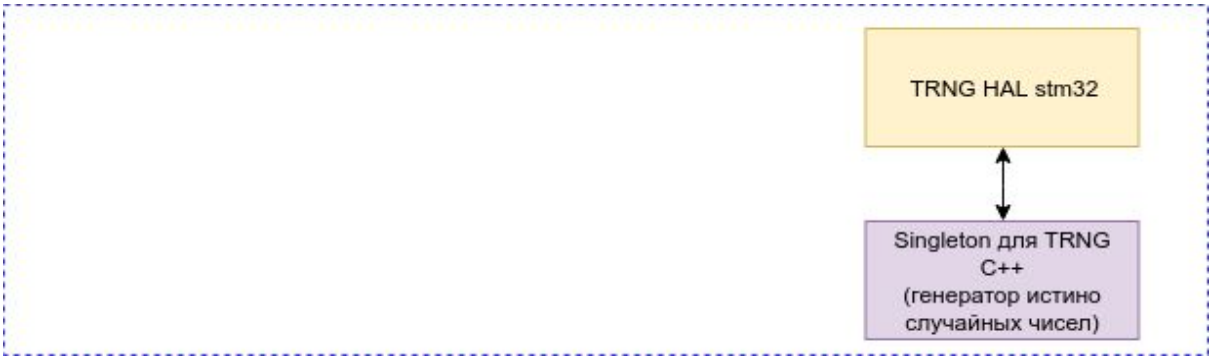

Рисунок 6. Внутреннее представление взаимодействия с TRNG.

8. Уровень взаимодействия для работы с уникальным идентификатором:
   a. Нижний уровень взаимодействия представлен через паттерн проектирования Singleton. Применяется для разделения уровня HAL UID STM32 и уровня взаимодействия с бизнес логикой (рисунок 7).

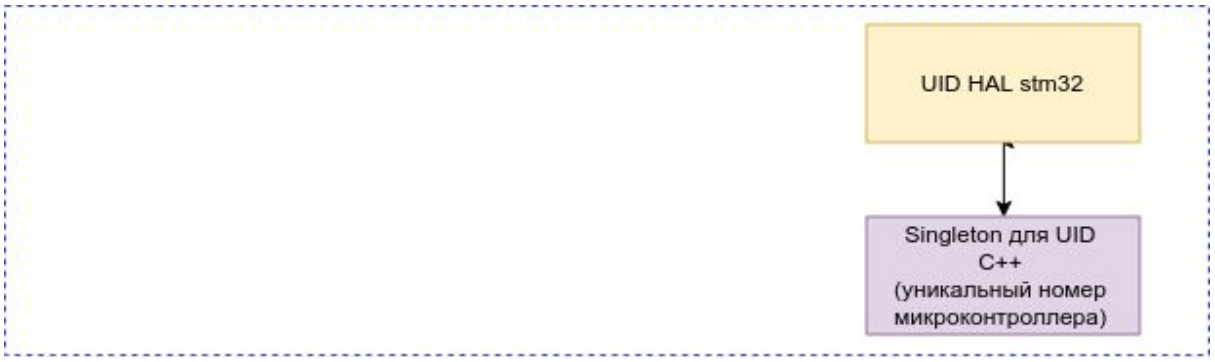

Рисунок 7. Внутреннее представление взаимодействия с UID.

8. Уровень взаимодействия для работы с GPIO:
    a. Нижний уровень взаимодействия представлен через паттерн проектирования Singleton. Применяется для разделения уровня HAL GPIO STM32 и уровня взаимодействия с бизнес логикой. Данный уровень взаимодействия с GPIO необходим для получения конфигурации системы, работа с кнопкой и светодиодной индикацией, а также взаимодействие с кнопкой увеличения громкости (рисунок 8).

Рисунок 8. Внутреннее представление взаимодействия с GPIO.

9. Уровень отладки и тестирование комплекса целиком:
    a. Отладка операционной системы реального времени FreeRTOS происходит через Tracealyzer (TraceRecorder для FreeRTOS). Для отладки используется streamports/TCPIP. TraceRecorder поддерживает большое количество интерфейсов через которые можно получать отладочную информацию для Tracealyzer, а так же Tracealyzer поддерживает множество других операционных системы (рисунок 9).
    b. Отладочные сообщения отправляются с помощью механизма Instrumentation Trace Macrocell (ITM) ARM.

Рисунок 9. Внутреннее представление взаимодействия с механизмом отладки и тестированием системы.

10. Уровень взаимодействия с FreeRTOS:
    a. Взаимодействие с FreeRTOS происходит через общий интерфейс CMSIS-RTOS API, код приложения взаимодействия с CMSIS-RTOS, применение CMSIS-RTOS позволяет более унифицировать сам метод работы с ОС реальными времени, а так же создает унифицированный интерфейс, который позволяет использовать иные ОС реального времени (не FreeRTOS). Уровень разделения функционала и API для большинства RTOS, необходим при замене и переходе на иную от текущей RTOS (рисунок 10).
    b. Для плат СТ, СЦ, СЛ используется методы выделения памяти и работы с памятью heap_4 и heap_5. Метод heap_5 используется только платы которая имеет внешнюю оперативную память.
    c. CMSIS RTOS как общий интерфейс взаимодействия с бизнес логикой.

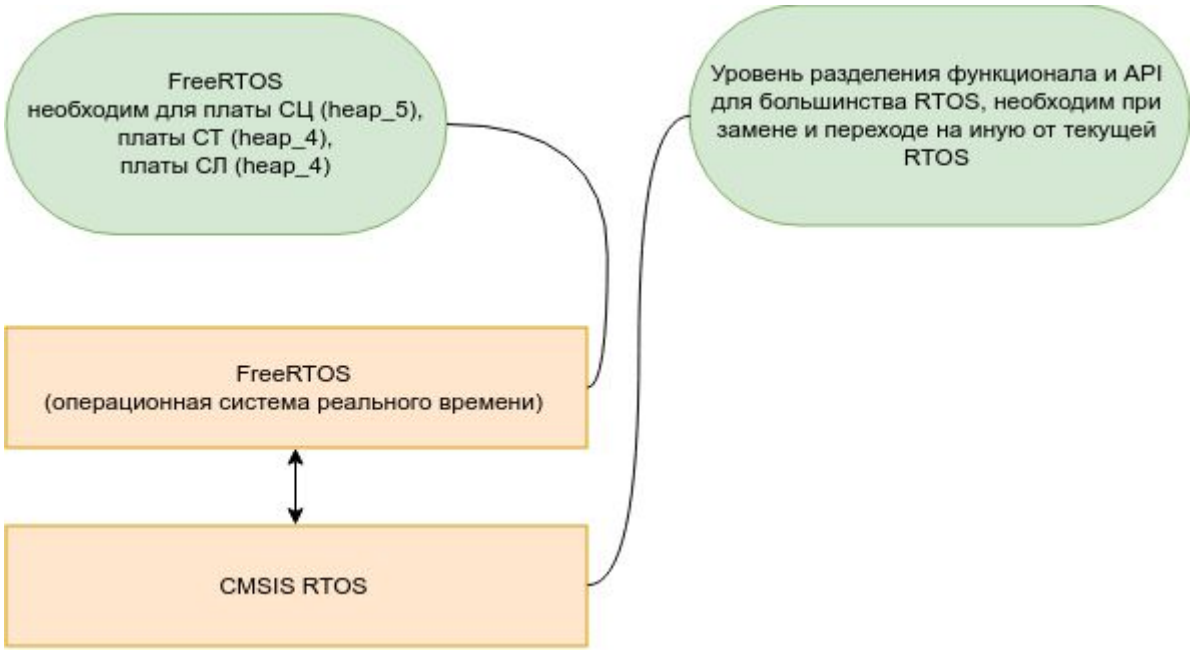

Рисунок 10. Внутреннее представление взаимодействия с механизмом FreeRTOS/CMSIS RTOS.

11. Уровень взаимодействия для работы со сторожевым таймером (WDT):
    a. Нижний уровень взаимодействия представлен через паттерн проектирования Singleton. Применяется для разделения уровня HAL WDT STM32 и уровня взаимодействия с бизнес логикой (рисунок 11).

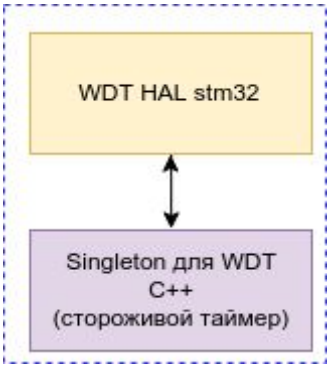

Рисунок 11. Внутреннее представление взаимодействия с механизмом WDT.

12. Уровень взаимодействия для работы с LwIP и mbedTLS:
    a. Уровень передачи данных и взаимодействия по Ethernet происходит благодаря библиотеке LwIP (рисунок 12). Настройка IGMP.
    b. Библиотека mbedTLS распространяется под лицензией Apache 2. Библиотека mbedTLS содержит алгоритмы шифрования. Модули библиотеки реализованы независимо друг от друга, что позволяет выполнить гибкую конфигурацию библиотеки mbed TLS. Для конфигурирования используется конфигурационный заголовочный файл mbedtls_config.h. MbedTLS используется построения

защищенного соединения между HTTPS – сервером и клиентом, а так же шифрует все голосовые сообщение передавыемые с помощью RPT.

c. HTTPS выбран базовым протоколом функционального взаимодействия всей системы целиком, а именно получение статусов работы, проверка принадлежность к системы, отправка файлов и т.п. Так же структура и синтаксис сообщений SIP идентичны используемым в протоколе HTTP.
d. Для синхронизации внутреннего времени микроконтроллера используется протокол SNTP/NTP.
e. Отправка данных осуществляется только для ПО “Конфигуратор” и ПО “ПДКВ”.

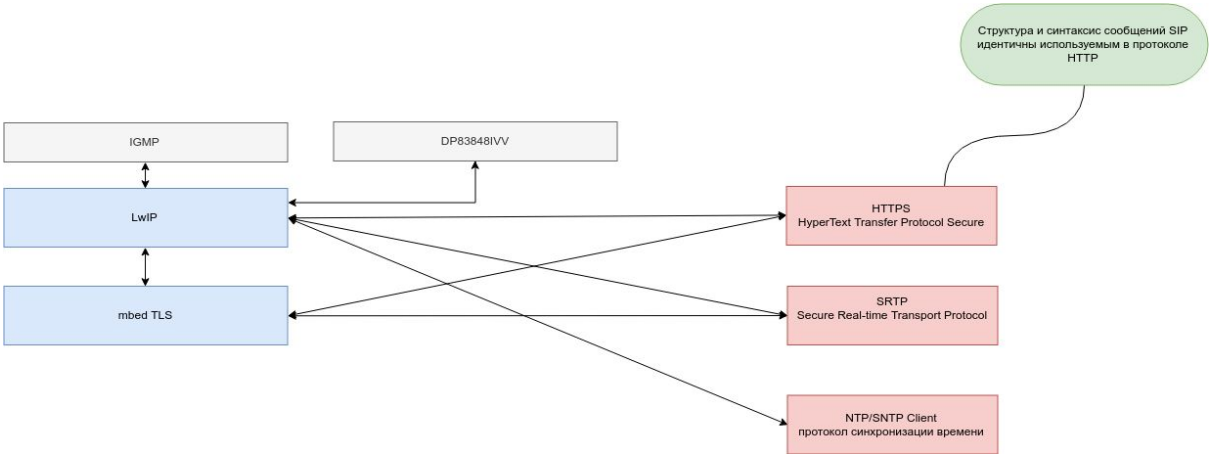

Рисунок 12. Уровень взаимодействия для работы с LwIP и mbedTLS.

13. Уровень конфигурации через JSON (рисунок 13):
    a. Функционал который передаете в JSON формате:
        i. Запрос абонентскому устройству о статусе (онлайн, готов, занят, оффлайн). Оффлайн-статус определяется по заданному таймауту.
        ii. Проверка принадлежности устройства системе “Git-Comm IPS”.
        iii. Передача коммутационного плана/файла прошивки.
        iv. Отправка сообщения в устройство логирования.
        v. Команда на установку выхода/выходов.
        vi. Запрос статуса входов.
        vii. Команда на перезагрузку коммутационного плана.
        viii. Команда на перезагрузку устройства.
        ix. Проверка обновления ПО устройства.
        x. Серия запрос/ответ на установку, синхронизацию и завершение симплексной связи и дуплексной связи.
        xi. Серия запрос/ответ на установку, синхронизацию и завершение.
        xii. Создание конференции, добавление абонента в конференцию, удаление абонента из конференции, удаление конференции.
        xiii. Конфигурация/настройка TLV320AIC3254.

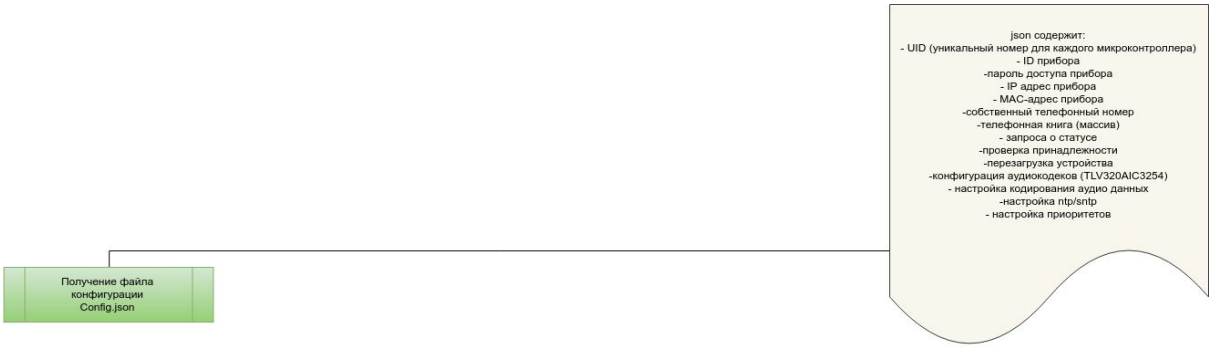

Рисунок 13. Уровень конфигурации через JSON.

Пример сообщения получения статуса:

```
{
        "type":"status",
        "code":0,
        "v_software":"s0",
        "v_communication":"c0"
}
```

Ключ	**Тип**	**Описание**
type	строка	Тип сообщения
code	целое число	Код устройства о статусе (онлайн, готов, занят)
v_software	строка	Версия ПО
v_communication	строка	Версия коммутационного плана

Пример сообщения проверки принадлежности устройства “Git-Comm IPS”:

```
{
        "type":"own",
        "uid":"123345677AAAA",
        "check":"ok",
        "a_communication": "plan 0"
}
```

Ключ	**Тип**	**Описание**

type	строка	Проверка принадлежности устройства
uid	строка	Уникальный номер устройства
check	строка	Проверка корректности настройки
a_communication	строка	Актуальный коммутационный план

Пример сообщения проверки наличия коммутационного плана и файла прошивки:

```
{
	"type":"file",
	"file":"firmware",
	"file_availability":"ok"
}
```

Ключ	**Тип**	**Описание**
type	строка	Тип сообщения
file	строка	Файл прошивки или файл коммутационного плана
file_availability	строка	Наличие файла

Пример сообщения логирования

```
{
	"type":"log",
	"message":"Устройство работает стабильно"
}
```

Ключ	**Тип**	**Описание**
type	строка	Тип сообщения

message	строка	Отправка сообщения логирования

Пример сообщения установки выхода/входа

```
{
	"type":"input/output",
	"setup": [
		{
			"pin": 0
			"state": "input"
			"value": 1
		},
		{
			"pin": 1
			"state": "output"
			"value": 0
		},
		...
	]
}
```

Ключ	**Тип**	**Описание**
type	строка	Тип сообщения
setup	массив	Массив значений на установку пинов
pin	целое число	Номер пина на установку
state	строка	Установка пина вход или выход
value	целое число	Значение на установку

Пример сообщения статусов входов:

```
{
	"type":"status pins",
	"state": [
```

```
                {
                        "pin": 0
                        "state": "input"
                        "value": 1
                },
                {
                        "pin": 1
                        "state": "input"
                        "value": 0
                },
                …
        ]
}
```

Ключ	**Тип**	**Описание**
type	строка	Тип сообщения
state	массив	Массив значений на получение статусов пинов
pin	целое число	Номер пина на установку
state	строка	Установка пина вход или выход (чтение установленного типа)
value	целое число	Значение на установку

Пример сообщения на перегрузку коммутационного плана или устройства

```
{
        "type":"reset",
        "cmd":"plan"
}
```

Ключ	**Тип**	**Описание**
type	строка	Тип сообщения

cmd	строка	Перегрузка коммутационного плана или устройства

Пример сообщения на перегрузку коммутационного плана или устройства

```
{
        "type":"connection",
        "cmd":"request/response",
        "link":"simplex",
        "connect":"on/off"
}
```

Ключ	**Тип**	**Описание**
type	строка	Тип сообщения
cmd	строка	Запрос или ответ
link	строка	Симплексная ил дуплексная связь
connect	строка	Установка и завершение связи

Пример сообщения на создание/добавление/удаление абонента и конференции.

```
{
        "type":"conference",
        "cmd":"open/add/close",
        "sub":"subscriber 1"
}
```

Ключ	**Тип**	**Описание**
type	строка	Тип сообщения
cmd	строка	Создание/добавление/удаление абонента и конференции
sub	строка	Абонент

Пример сообщения на установку и получении настроек TLV320AIC3254

```
{
	"type":"TLV320AIC3254",
	"cmd":"request/response",
	"channel":0,
	"input_gain":0,
	"agc": {
		"status":"off",
		"max_level":0,
		"max_gain":40,
		"response_time":20,
		"recovery_time":400
	},
}
```

Ключ	**Тип**	**Описание**
type	строка	Тип сообщения
cmd	строка	Запрос/Ответ
channel	целое число	Номер канал
input_gain	целое число	Усиление громкости
agc		Автоматическое усиление сигнала
status	строка	Включен/выключен параметр автоматического усиление сигнала
max_level	целое число	Максимальное уровень сигнала
max_gain	целое число	Коэффициент усиления

response_time	целое число	Время отклика АРУЗ
recovery_time	целое число	Время восстановления АРУЗ

Пример сообщения на установку и конфигурацию прибора


```
{
	"type":"config",
	"cmd":"request/response",
	"id":0,
	"password":"12345678",
	"ip":"192.168.0.10",
	"mac":"00:0a:95:9d:68:16",
	"phone_number":100
}
```


Ключ	**Тип**	**Описание**
type	строка	Тип сообщения
cmd	строка	Запрос/Ответ
id	целое число	ID прибора
password	строка	Пароль доступа
ip	строка	IP-адрес прибора
mac	строка	MAC адрес прибора
phone_number	целое число	телефонный номер прибора

В процессе разработке поля JSON могут поменять свое название, иметь незначительные изменения, а так же могут появится новые поля которые не были учтены при разработке архитектуры ВПО, все изменения структуры JSON полей согласуются с заказчиком.

14. Описание бизнес логики:
    a. Разработка программного обеспечения для бизнес логики происходит на языке программирования C++.
    b. При разработке используется свободный кодек для сжатия речевого сигнала – Speex.
    c. Логирование заносятся в кольцевой буфер (кольцевой буфер эффективных способов организовать FIFO без использования динамической памяти).
    d. Создание общего (обобщенного) интерфейса для json через паттерн Facade (определяет унифицированный высокоуровневый интерфейс к подсистеме json, что упрощает использование при передачи через rs232 и ethernet). Facade "обертывает" эту подсистему более простым интерфейсом.
    e. Применение паттерна State для определения поведения при изменении вида связи и режимов работы. Паттерн State позволяет объекту изменять свое поведение в зависимости от внутреннего состояния. Создается впечатление, что объект изменил свой класс. Паттерн State является объектно-ориентированной реализацией конечного автомата. Так же паттерном определяются поведенческие свойства световой индикации.
        i. Прямая связь (ПС)
        ii. Групповая связь (ГС)
        iii. Циркулярная связь (ЦС)
        iv. Конференц-связь (КС)
        v. Телефонная связь (ТС)
    f. Бизнес логика определяет работу задач (потоков) FreeRTOS в контексте CMSIS RTOS.
    g. Применение паттерна Observer при изменении конфигурации json. Паттерн Observer определяет зависимость "один-ко-многим" между объектами так, что при изменении состояния одного объекта все зависящие от него объекты уведомляются и обновляются автоматически. Observer инкапсулирует главный (независимый) компонент в абстракцию Subject и изменяемые (зависимые) компоненты в иерархию Observer.

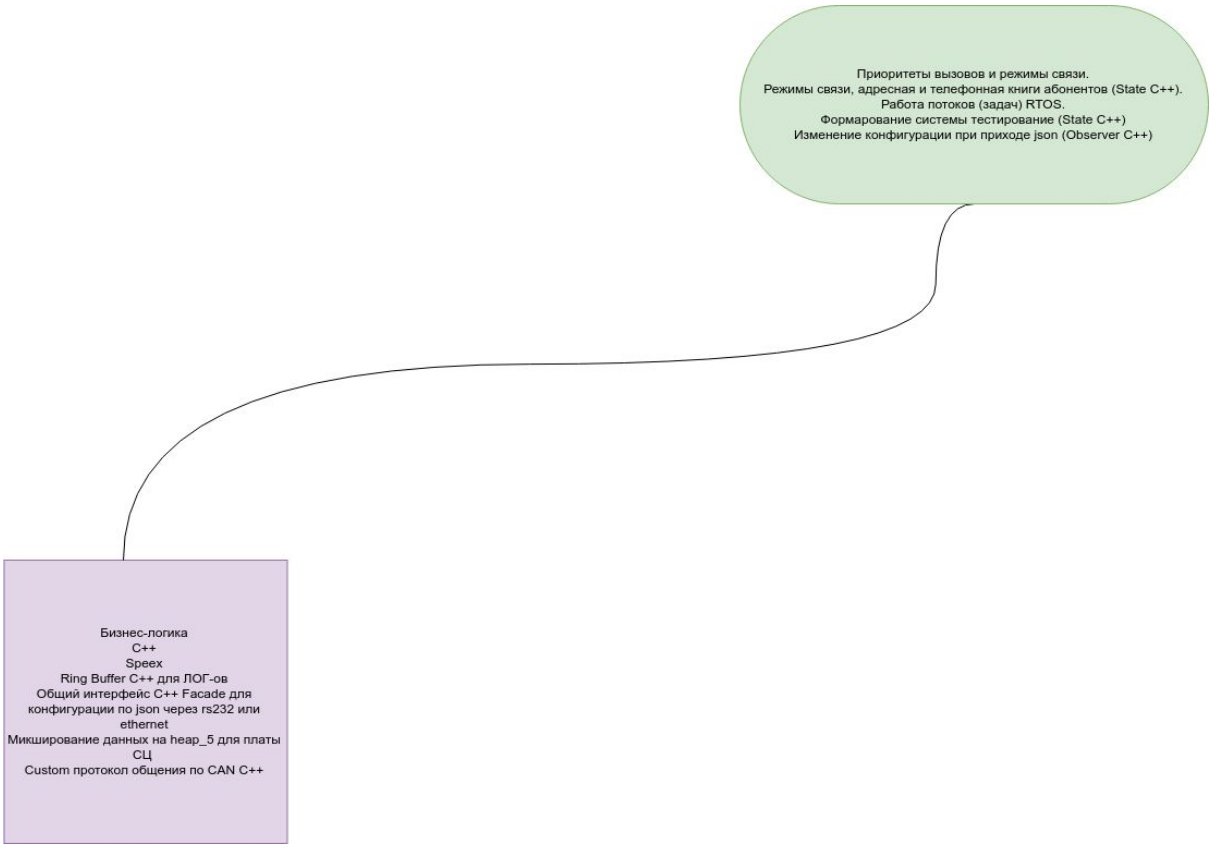

Рисунок 14. Описание бизнес логики.

15. Правила применяемые при разработке встраиваемого программного обеспечения:
    a. При разработке программного обеспечения код будет комментироваться с помощью doxygen (Qt style).
    b. Форматирование кода: clang-format (стиль LLVM).
    c. Анализ кода: Clang-Tidy, Clazy, cppcheck.
    d. Данные правильно применяются ко всему программному коду.

Архитектура встраиваемого программного обеспечения для плат УИ.

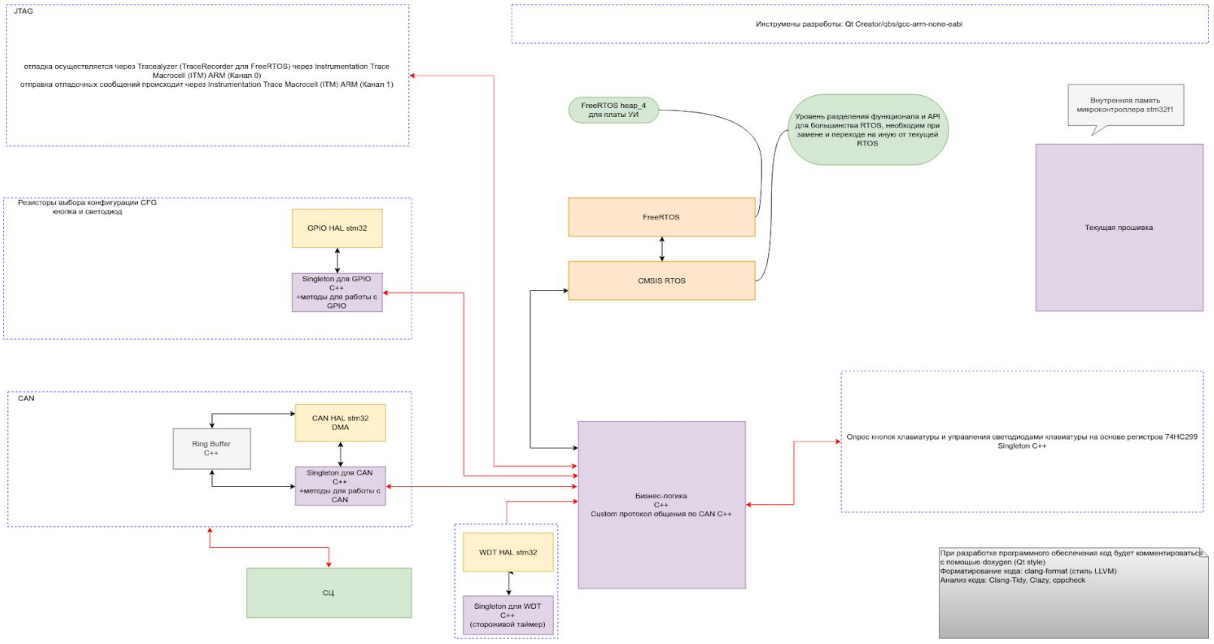

Рисунок 15. Архитектура ВПО (встраиваемого программного обеспечения) для плат УИ.

Архитектура ВПО для плат УИ заимствует программные блоги из архитектуры на платы СЦ, СТ, СЛ: CAN класс, GPIO класс, WDT класс, а так же взаимодействие через FreeRTOS в контексте CMSIS RTOS.

Отличительные черты архитектуры плат УИ:

1. Система тестирование и отладки плат УИ.
    a. Отладка операционной системы реального времени FreeRTOS происходит через Tracealyzer (TraceRecorder для FreeRTOS). Для отладки используется streamports/ARM_ITM. TraceRecorder поддерживает большое количество интерфейсов через которые можно получать отладочную информацию для Tracealyzer, а так же Tracealyzer поддерживает множество других операционных системы (рисунок 16).
    b. Отладочные сообщения отправляются с помощью механизма Instrumentation Trace Macrocell (ITM) ARM.
    c. Для отладки RTOS настраивается канал 0, для получения отладочных сообщений бизнес логики настраивается канал 1.

Рисунок 16. Система тестирование и отладки плат УИ.

2. Система опроса кнопок и установки состояний светодиодов.
   a. Нижний уровень взаимодействия представлен через паттерн проектирования Singleton. Применяется для разделения уровня HAL GPIO STM32 и уровня взаимодействия с бизнес логикой. Данный уровень взаимодействия с GPIO необходим для установки поведения системы световой индикации, а так же получение событий от кнопок при формировании соответствующего действия. (рисунок 17).

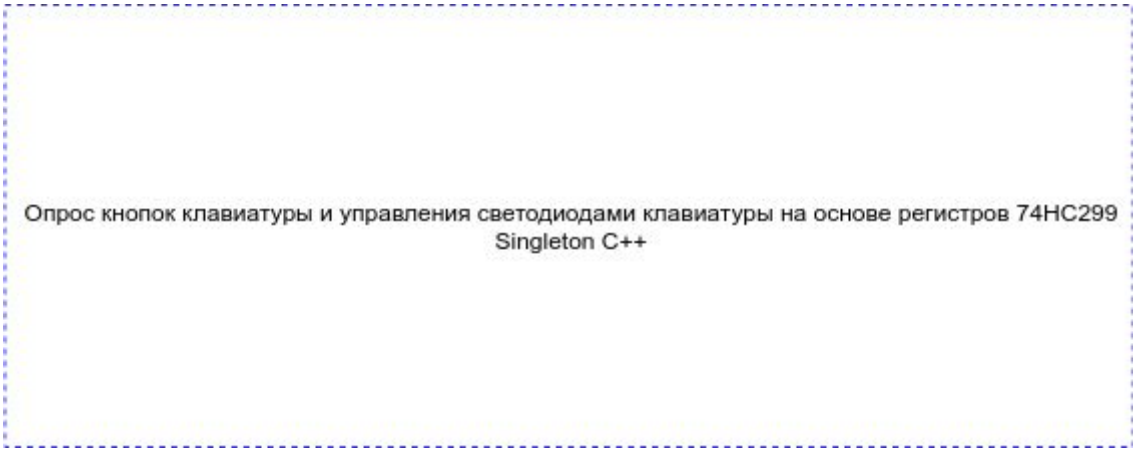

Рисунок 17. Система опроса кнопок и установки состояний светодиодов.

Архитектура программного обеспечения “Конфигуратор” и программного обеспечения логирования “ПДКВ”

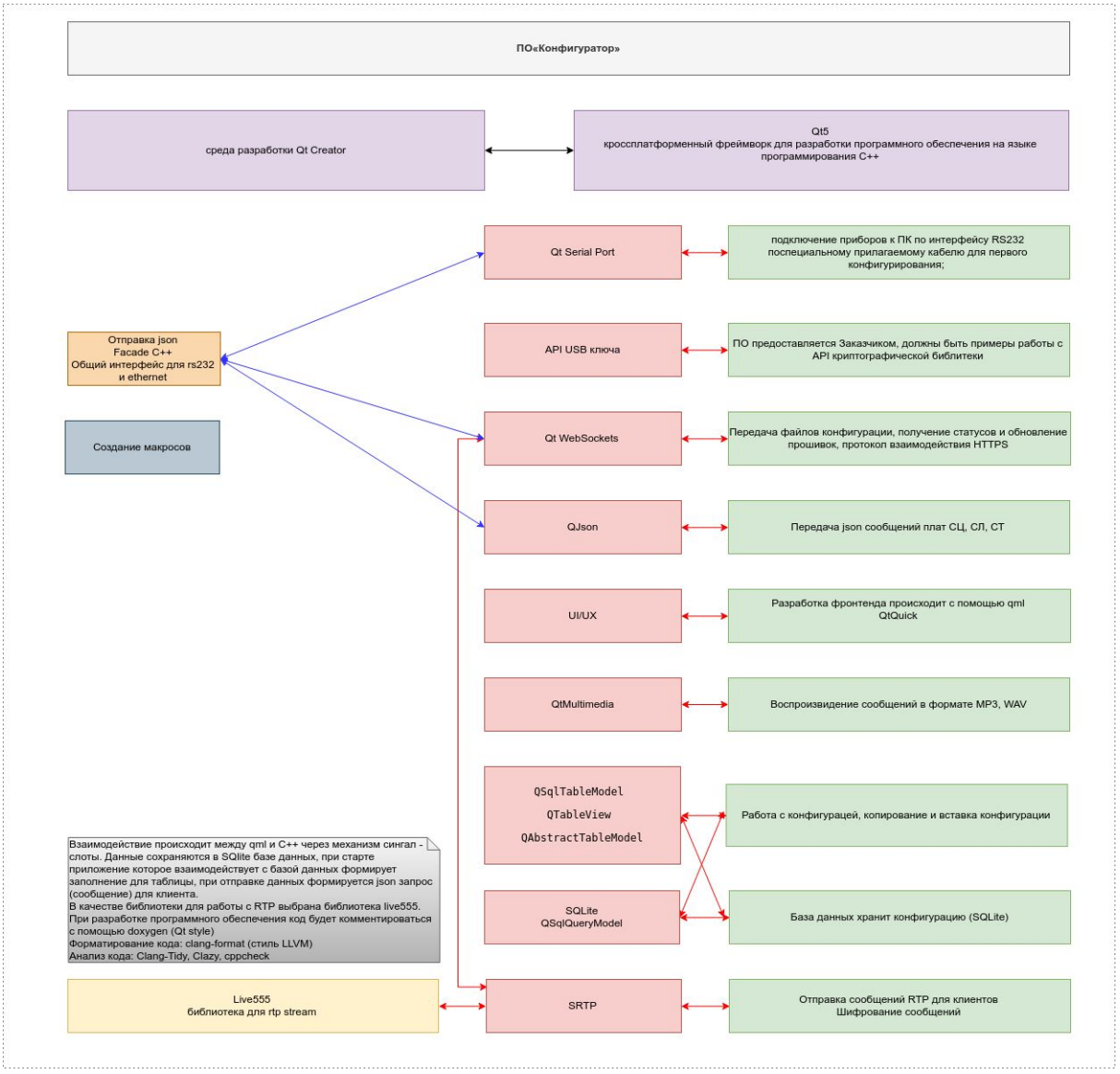

Рисунок 18. Структура ПО “Конфигуратор”

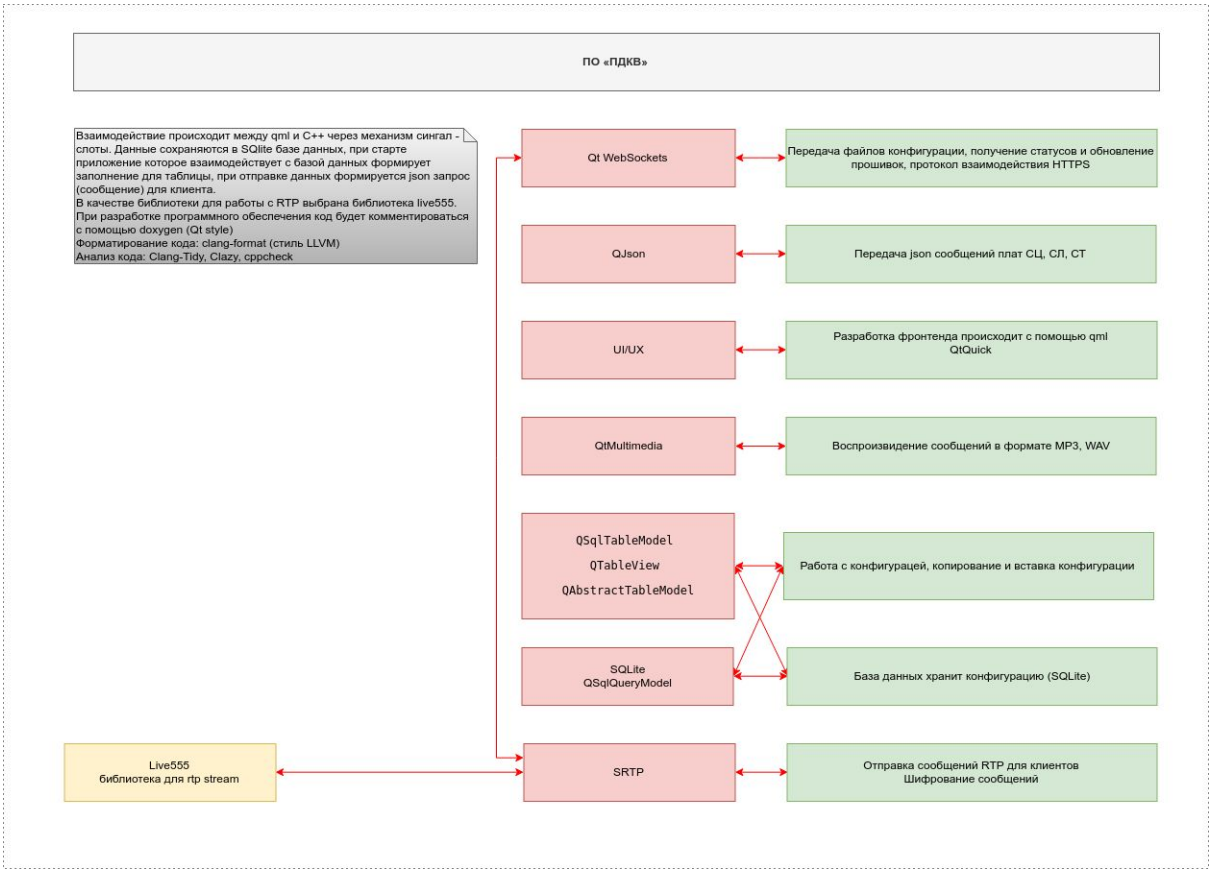

Рисунок 19. Структура ПО “ПДКВ”

ПО “Конфигуратор” и “ПДКВ” обладают идентичными программными блоками и компонентами, поэтому описание для них будет осуществляться в комплексе.

1. Программирование и конфигурирование приборов через RS232. Работа с портом rs232 осуществляется с помощью модуля Qt Serial Port, при работе с данным модулем используется классы QSerialPort и QSerialPortInfo. Для взаимодействия с UI частью программного обеспечения используется механизм сигна-слотов, механизм сигна-слотов используется для взаимодействие между qml - c++ частям кода. Qml язык описания UI/UX часть интерфейса пользователя, QML — это язык разметки для создания пользовательских интерфейсов. Его основная задача — обеспечить возможность простого и быстрого создания приложений с красивым, анимированным интерфейсом. QML - декларативный язык программирования, основанный на JavaScript.
2. Взаимодействие с API USB ключа, происходит только при старте ПО “Конфигуратор”.
3. При передаче файлов конфигурации, получении статусов и обновление прошивок, используется протокол взаимодействия HTTPS. Для взаимодействия по HTTPS на стороне QT используется модуль Qt WebSockets. При работе с данным модулем используется класс QtWebSockets, а также модуль QtNetwork (QSslCertificate и QSslKey).

4. Для обмена и создания JSON сообщений используется модуль QJson (QJsonObject)
5. Разработка фронтенд часть происходит с помощью qml и модуля QtQuick.
6. Для воспроизведения мультимедийной (аудио) информации используется модуль Qt Multimedia, а так же классы QAudioDeviceInfo, QAudioInput для детектирования и вопроизвдения мультимедийной для конкретной аппаратуры.
7. Для работы с полями конфигурации, с копирование и переносом информации используется следующие элементы разработки: QSqlTableModel, QTableView и QAbstractTableModel.
8. Вся информация по конфигурации сохраняется в базе данных, SQlite основная база данных которая выбрана для проекта, для взаимодействия с данной базой данных используется следующие элементы: SQLite и QSqlQueryModel.
9. Для отправки и получения данных по RTP используется библиотека Live555. Live555 это (LGPL) C++ библиотека, данная библиотека используется в таких продуктах, как VLC и mplayer.
10. Аудиоданные (переговоры) записываются на диск (файловую систему данной ОС), в базе данных сохраняется только путь к файлу на диске.
11. Создание общего (обобщенного) интерфейса для json через паттерн Facade (определяет унифицированный высокоуровневый интерфейс к подсистеме json, что упрощает использование при передачи через rs232 и ethernet). Facade "обертывает" эту подсистему более простым интерфейсом.
12. Версия Qt от 5.12 и новее.