ООО «ГТЛАБ»

ПО «СТРИММЕР»

Руководство системного программиста

Саров 2025

Оглавление

1. Общие сведения о программе 3
2. Структура программы 4
3. Настройка программы 6
3.1 Общий раздел конфигурационного файла 7
3.2 Конфигурационный раздел устройств 9
3.2.1 Настройка вибро-контроллера 11
3.2.2 Настройка генератора 13
3.3 Конфигурационный раздел слотов 14
3.4 Примеры конфигурационных файлов 16
4. Отказоустойчивый режим работы 20
5. Проверка программы 21
6. Запуск Docker -контейнера 23
7. Интеграция с разрабатываемыми приложениями 25
7.1 Методы мониторинга состояний 28
7.1.1 Метод slotState 29
7.1.2 Метод stackState 31
7.1.3 Метод connectionsState 33
7.2 Использование интерфейса PUBLISHER 34
Источники, использованные при разработке 36

1. Общие сведения о программе

ПО «Стриммер» (от англ. stream – поток, streamer – в нашем случае подразумевается: передающий поток данных) представляет собой консольную утилиту, предназначенную для трансформации протокола ADCCluster (протокол обмена данными с АЦП производства ООО «ГТЛАБ» серии D0XXX) в простой поток данных, передаваемых с помощью TCP-соединения.

ПО «Стриммер» используется в качестве промежуточного преобразователя данных для организации передачи данных от АЦП в клиентское ПО для дальнейшей обработки. Клиентским ПО может являться ПО «GTL», ПО «GTLD-DESKTOP», ПО «GTLD-DAEMON» или другое программное обеспечение, созданное в соответствии с данным руководством.

ПО «Стриммер» совместимо с ОС Windows и Linux.

Предусмотрен также Docker-контейнер для запуска ПО «Стриммер».

Дистрибутивы ПО «Стриммер», Docker-контейнер с ПО «Стриммер», а также данное руководство, доступны для скачивания по ссылке:

https://downloads.gtlab.pro/files/streamer/

2. Структура программы

На рисунке 1 представлена структурная схема ПО «Стриммер».

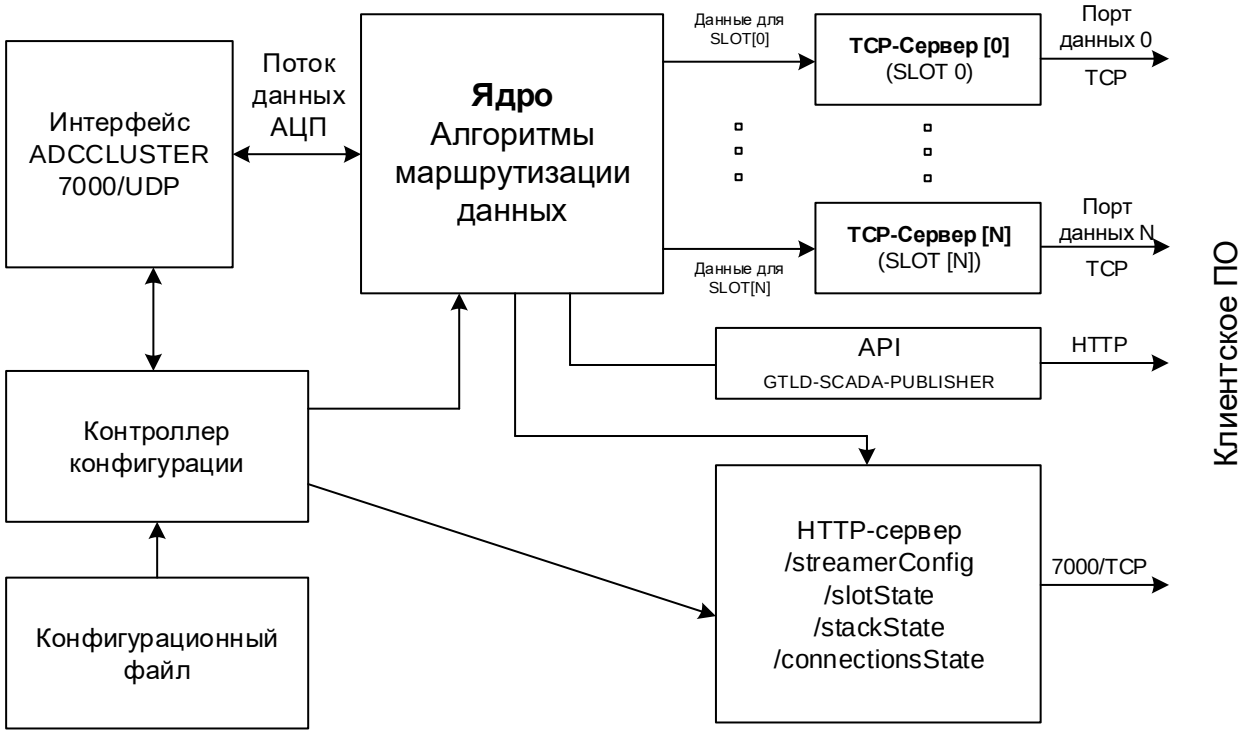

Рисунок 1 – Структурная схема ПО «Стриммер»

Ядро программы обеспечивает маршрутизацию потока данных от АЦП к серверам TCP-соединений. Данные маршрутизируются в зависимости от созданной пользователем конфигурации. Конфигурация определяет перечень входных портов АЦП, данные от которых будут перенаправлены на тот или иной TCP-сервер.

Замечание: *В документе для обозначения физических входов АЦП используется слово «**порт**», для обозначения логического выхода из ПО «Стриммер» используется слово «**канал**». Эти обозначения перекликаются, как с терминологией конфигурационных файлов, так в ключах объектов, характеризующих состояние ПО, и во многих других случаях.*

Для обозначения совокупности выходных каналов и их соответствие входным портам АЦП используется термин «слот» (SLOT). Другими словами, слоты определяют связь между выходными каналами ПО и входными портами АЦП. Один экземпляр ПО может обслуживать произвольное количество слотов, каждый слот может содержать произвольное число каналов, каждый канал прикреплён к одному порту АЦП, при этом в любом слоте могут применяться каналы, использующие одни и те же входные порты АЦП.

Каждому слоту соответствует один TCP-сервер и перечень входных портов АЦП, данные от которых будут передаваться через него. Порядок следования каналов в

конфигурации слота имеет значение – в таком же порядке данные будут размещены в TCP-трафике. Конфигурация слотов определяется соответствующим конфигурационным файлом.

Для корректной работы клиентского ПО оно должно выполнить чтение текущей конфигурации ПО путем выполнения HTTP-GET запроса на адрес «/streamerConfig» – для этого в состав ПО включен HTTP-сервер. Получив текущую конфигурацию клиентское ПО может осуществить подключение к нужному TCP-серверу и вести обработку получаемых данных. Кроме этого, для мониторинга состояния ПО и АЦП, подключенных к нему, предусмотрены три HTTP-метода: «/slotState», «/stackState» и «/connectionsState». Подробнее описание данных методов и остальные вопросы интеграции ПО в разрабатываемое клиентское приложение рассмотрен в разделе 7.

ПО оснащено поддержкой API взаимодействия с GTLD-SCADA (интерфейс PUBLISHER). Это позволяет осуществлять взаимодействие с внешним сервисами, путём передачи им с заданной периодичностью состояния ПО, а также асинхронную передачу сообщений о событиях, возникающих в процессе работы ПО. К таким событиям относятся: сбои в работе АЦП, изменение состояний портов АЦП, подключение и отключение клиентских приложений и т.п.

Подробнее особенности работы API изложены в соответствующей документации [1, 2]. Этот API также можно использовать для интеграции ПО в разрабатываемое приложение.

3. Настройка программы

Конфигурирование ПО выполняется с помощью конфигурационного файла.

При запуске ПО путь к конфигурационному файлу можно передать через аргумент командной строки, как показано на рисунке 2 или через переменную окружения **CONFIG_FILENAME**.

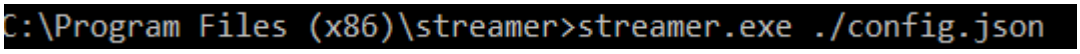

Рисунок 2 – Запуск ПО с конфигурационным файлом

Возможен также запуск ПО без использования конфигурационного файла. Запуск без конфигурационного файла приведет к следующему варианту поведения (поведение по умолчанию): контроллер конфигурации выполнит попытку соединения с первым найденным в сети АЦП, выполнит чтение его конфигурации и создаст единственный слот в котором будут все порты найденного АЦП, каждому порту АЦП будет создан соответствующий канал. Для этого слота программа запустит один TCP-сервер и будет выполнять передачу данных со всех портов АЦП во все клиентские соединения, которые будут подключаться к TCP-серверу. Интерфейс PUBLISHER задействован не будет. Сбои АЦП будут приводить к немедленному завершению работы ПО.

Конфигурационный файл включает в себя следующие разделы:

- **Общий раздел**. Конфигурация основных параметров работы ПО;
- **Раздел устройств**. Список устройств, настройки портов АЦП, режимов работы встроенного генератора и вибро-контроллера, если таковые поддерживаются устройством;
- **Раздел слотов**. Конфигурация выходных каналов ПО, определяющая маршрутизацию данных, поступающих из АЦП.

Конфигурационный файл представляет собой текстовый файл в котором в формате JSON описана конфигурация ПО. Ключи общего раздела являются ключами первого уровня, поддерживаемые настройки общего раздела представлены в разделе 3.1.

Раздел конфигурирования устройств размещается в корневом ключе **units** и является массивом объектов настройки устройств, подробнее его структура раскрыта в разделе 3.2.

Раздел конфигурирования слотов размещается в корневом ключе **slots** и является массивом объектов настройки слотов, подробнее его структура раскрыта в разделе 3.3.

3.1 Общий раздел конфигурационного файла

В таблице 1 представлены ключи общего раздела конфигурационного файла.

Таблица 1 – Ключи общего раздела конфигурационного файла

Ключ параметра	**Тип значения**	**Диапазон**	**Комментарий**
mod_id	String	String	Идентификатор модуля
httpPort	Int	65535	Порт http-интерфейса
initTime	Int	0-100000	Время инициализации стека АЦП, мс
defaultValue	Float	Float	Транслируемое при сбое значение
controllerElapsedAlert	Int	0-100	Контроль задержки основного цикла, мс
fatalFail	Bool	True	Останавливать ПО при сбоях АЦП
		False	Продолжать работу при сбоях АЦП
fixedMaster	Bool	True	Запрещено автоматически менять мастера
		False	Разрешено автоматически менять мастера
publisherURL	String	String	URL HTTP-интерфейса PUBLISHER [1]
publisherIntervalMs	Int	0-10000	Интервал передачи объекта состояний, мс [1]
stateRouteKey	String	String	Маршрутный ключ состояний [1]
eventsRouteKey	String	String	Маршрутный ключ событий [1]
units	**[]**	**[]**	**Раздел устройств (п. 3.2)**
slots	**[]**	**[]**	**Раздел слотов (п. 3.3)**

- **mod_id -** в данном параметре указывается уникальный идентификатор экземпляра ПО. В случае использования в системе нескольких экземпляров ПО целесообразно использование данного параметра в целях разделения сообщений о событиях полученных от различных экземпляров. Также этот параметр выдается через HTTP-метод streamerConfig;
- **httpPort -** номер порта HTTP-сервера, по умолчанию используется порт 7000/TCP, изменение данного параметра может быть востребовано при запуске нескольких копий ПО на одном хосту;
- **initTime -** время ожидания инициализации АЦП. По истечении данного времени стек будет проинициализирован с теми АЦП, которые успели выйти на связь, актуально при старте ПО;
- **defaultValue -** значение, которое будет транслироваться в канал в том случае если заданное в конфигурации слота АЦП не вышло на связь либо, связь с ним была потеряна в процессе работы;
- **controllerElapsedAlert -** Диагностический параметр, применяется для оценки времени выполнения основного цикла маршрутизации данных. Целесообразно применять для оценки производительности ПО в условия повышенной нагрузки. При значительном количестве клиентов время выполнения основного маршрутного цикла увеличивается и если оно превысит заданное в данном параметре, ПО выдаст в STDOUT предупреждающее сообщение. Не рекомендуется превышать время цикла 50мс;

- **fatalFail -** при установки данного параметра в true, в целях предотвращения передачи недостоверных данных, ПО прекратит работу в том случае, если потеряна связь с каким-либо АЦП;
- **fixedMaster -** в зависимости от схемотехнических решений, различные модели АЦП позволяют произвольным образом выбирать мастера для работы в стеке, без изменения коммутации синхронизирующего сигнала. В зависимости от модели используемых АЦП иногда следует остановить работу стека, если потеряна связь с мастером (fixedMaster = true), в случае если fixedMaster = false при потери мастера будет выполнена попытка назначить другого мастера (следующего в списке заданных АЦП);
- **publisherURL -** Строка, содержащая адрес интерфейса PUBLISHER [1];
- **publisherIntervalMs -** Интервал передачи состояний ПО в интерфейс PUBLISHER [1];
- **stateRouteKey -** Маршрутный ключ объекта состояний [1];
- **eventsRouteKey -** Маршрутный ключ для событий [1].
- **units –** ключ для размещения конфигурации устройств;
- **slots –** ключ для размещения конфигурации слотов.

3.2 Конфигурационный раздел устройств

Конфигурация устройств задаётся в конфигурационном файле через ключ **units**.

Конфигурационный раздел устройств представляет собой массив объектов конфигурации устройств, в массиве может содержаться произвольное число устройств, они должны быть совместимы между собой по следующим параметрам:

- частота дискретизации АЦП;
- количество портов;
- наличие тахо-сигнала;
- разрядность АЦП.

Порядок следования устройств имеет значение, в указанном порядке будут назначаться и сменяться при необходимости ведущие АЦП. Массив доступных портов АЦП также будет формироваться, исходя из порядка следования АЦП в конфигурационном массиве.

В таблице 2 представлены ключи раздела конфигурации устройств.

Таблица 2 – Ключи объекта конфигурации устройства

Ключ параметра	**Тип значения**	**Диапазон**	**Комментарий**
id	String	String	Идентификатор АЦП (серийный номер или IP)
portsConfig	[]	[]	Массив настроек портов
vibroConfig	{}	{}	Объект настройки вибро-контроллера
dacConfig	{}	{}	Объект настройки генератора

- **ids –** идентификатор устройства, это может быть его серийный номер или IP-адрес;
- **portConfigs –** содержит массив объектов конфигураций портов, при установки связи с АЦП каждый порт инициализируется в соответствии с настройками указанными в данных объектах. Порядковый номер следования конфигурационного объекта в массиве соответствуют номеру порта в АЦП. В объекте предусмотрен один параметр «mode», который может принимать значения: “AC”, “AC/DC” или “IEPE”;

Таблица 3 – Ключи объекта конфигурации портов

Ключ параметра	**Тип значения**	**Диапазон**	**Комментарий**
mode	String	AC	AC
		IEPE	IEPE
		AC/DC	AC/DC

- **vibroConfig –** содержит объект конфигурации вибро-контроллера, актуально для устройств поддерживающий функцию вибро-контроля, подробнее настройка вибро-контроллера представлена в разделе 3.2.1;

- **dacConfig –** содержит объект конфигурации генератора, актуально для устройств поддерживающий функцию генератора, подробнее настройка генератора представлена в разделе 3.2.2.

Пример простой конфигурации устройства без использования ключей **vibroConfig** и **dacConfig** представлен на рисунке 2.

Рисунок 2 – Простой пример конфигурации устройства

3.2.1 Настройка вибро-контроллера

Для использования вибро-контроллера в конфигурацию устройства следует добавить объект **vibroConfig**, пример такого объекта представлен на рисунке 3.

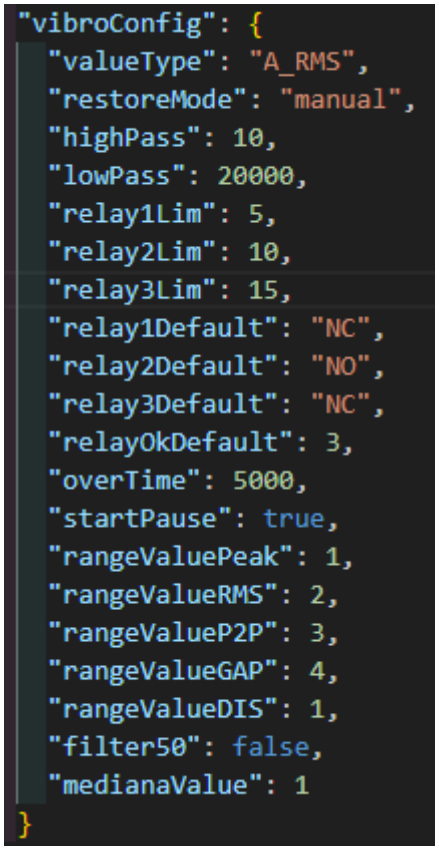

Рисунок 3 – Пример объекта vibroConfig

Не обязательно указывать все параметры, но все параметры, которые будут указаны, будут переданы устройству.

В таблице 4 представлено описание ключей объекта **vibroConfig**.

Таблица 4 – Ключи объекта конфигурации вибро-контроллера

Ключ параметра	**Тип значения**	**Диапазон**	**Комментарий**	
valueType	String	A_PEAK		
		A_RMS		
		A_P2P		
		V_PEAK		
		V_RMS		
		V_P2P		
		S_PEAK		
		S_RMS		
		S_P2P		
		STAT_DIS+24		
		STAT_DIS-24		
		R_VIB+24		
		R_VIB-24		
restoreMode	String	auto	Автоматический сброс реле	
		manual	Ручной сброс реле	
highPass	Float	TBD	Граница ФВЧ фильтра	
lowPass	Float	TBD	Граница ФНЧ фильтра	
relay1Lim	Float	TBD	Уровень включения реле 1	
relay2Lim	Float	TBD	Уровень включения реле 2	
relay3Lim	Float	TBD	Уровень включения реле 3	
relay1Default	String	NO	Нормально разомкнуто	
		NC	Нормально замкнуто	
relay2Default	String	NO	Нормально разомкнуто	
		NC	Нормально замкнуто	
relay3Default	String	NO	Нормально разомкнуто	
		NC	Нормально замкнуто	
relayOkDefault	int	0	Не используется	
		1	Реле 1	
		2	Реле 2	
		3	Реле 3	
overTime	int	[5:5000]	Время превышения порогового значения, кратно 5, (мс)	
startPause	bool	false	Старт без задержки	
		true	Задержка 20 секунд	
rangeValuePeak	int	0	14,1	A_PEAK V_PEAK S_PEAK
		1	28,2	
		2	70,7	
		3	141	
		4	282	
rangeValueRMS	int	0	10	A_RMS V_RMS S_RMS
		1	20	
		2	50	
		3	100	
		4	200	

3.2.2 Настройка генератора

Для использования генератора в конфигурацию устройства следует добавить объект **dacConfig**, пример такого объекта представлен на рисунке 4.

Рисунок 4 – Пример объекта dacConfig

Не обязательно указывать все параметры, но все параметры, которые будут указаны, будут переданы устройству.

В таблице 5 представлено описание ключей объекта **dacConfig**.

Таблица 5 – Ключи объекта конфигурации генератора

Ключ параметра	**Тип значения**	**Диапазон**	**Комментарий**
dacMode	int	0	Гармонический сигнал
		1	Шум
noiseHPF	double	[1; 10000]	Граница HP-фильтра для шума (Гц)
noiseLPF	double	[1; 50000]	Граница LP-фильтра для шума (Гц)
dacAmpl	int	[0; 5000]	Амплитуда (мВ)
dacFreq	double	[1; 10000]	Частота гармонического сигнала (Гц)
dacON	bool	false	DAC включен
		true	DAC выключен
stayON	bool	false	Выключить при отключении ПО «Стриммер»
		true	Оставаться включенным

3.3 Конфигурационный раздел слотов

Конфигурация слотов задаётся в конфигурационном файле через ключ **slots**.

Конфигурация слотов представляет собой массив объектов конфигурации слотов, каждый конфигурационный объект слота содержит два обязательных параметра: целочисленный параметр **dataPort** и массив конфигурационных объектов каналов - **channels**. Порядковый номер следования объектов конфигурации слотов имеет значение и называется **индексом слота**.

Таблица 6 – Ключи объекта конфигурации слотов

Ключ параметра	**Тип значения**	**Диапазон**	**Комментарий**
dataPort	Int	65535	TCP-порт сервиса передачи данных слота
channels	[]	[]	Массив настроек каналов

- **dataPort** - номер порта TCP, который будет принимать соединения и транслировать данные с входов АЦП указанных в массиве **channels**;
- **channels** – массив объектов конфигурации каналов. Порядок следования элементов массива **channels** определяет в какой последовательности данные будут размещены в TCP-трафике. Формат объекта конфигурации канала представлен в таблице 7.

Таблица 7 – Ключи объекта конфигурации канала

Ключ параметра	**Тип значения**	**Диапазон**	**Комментарий**
name	String	String	Имя канала
port	Int	65535	Номер порта АЦП
sensitivity	Float	Float	Чувствительность канала

- **name** – обязательный параметр с именем порта (используется для отображения в ПО «GTL», ПО «GTLD-DAEMON» и других);
- **port** - целочисленное значение, содержащее номер порта АЦП (если используется стек из нескольких АЦП, то port – это номер порта всего совокупного стека АЦП), соответствующее данному каналу;
- **sensitivity** – чувствительно канала (как правило это чувствительность датчика).

На рисунке 5 представлен пример конфигурации слотов.

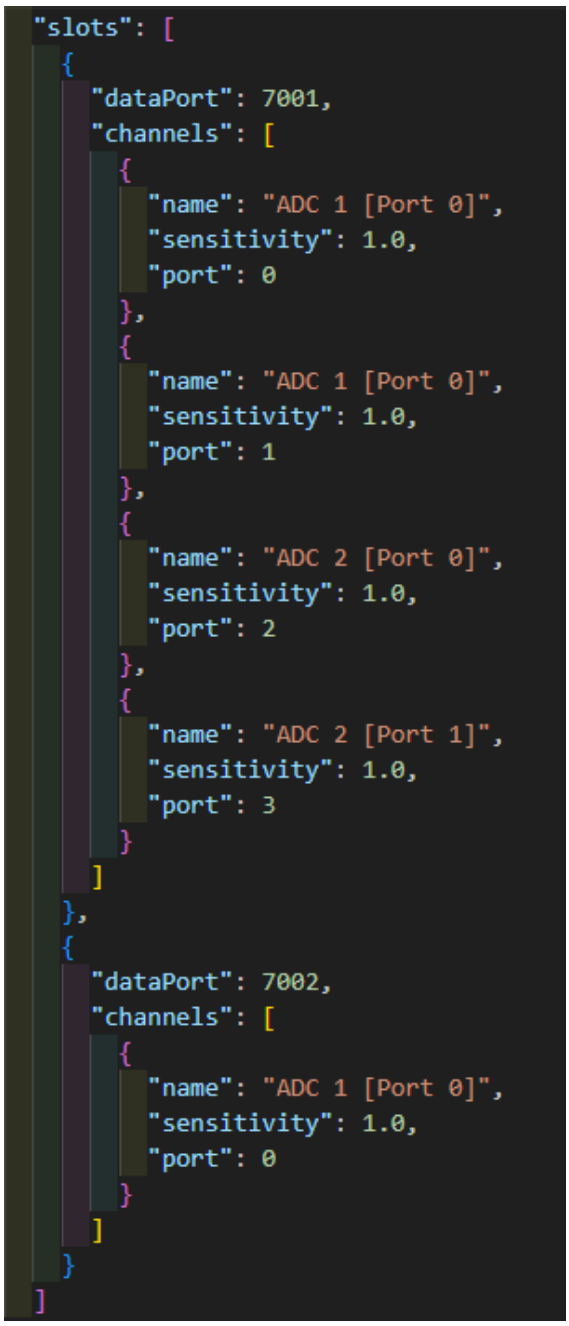

Рисунок 5 – Пример конфигурации слотов

В представленном примере подразумевается, что сформирован стек из двух двухканальных АЦП (ADC 1 и ADC 2). В этом случае общий стек АЦП содержит 4 порта с индексами от 0 до 3.

Конфигурационная структура, представленная на рисунке 5, формирует 2 слота: слот с индексом 0, который транслирует все 4 порта стека АЦП (ADC 1 + ADC 2) на TCP-порт **7001**, а слот с индексом 1 транслирует порт 0 первого АЦП (ADC 1) на TCP-порт **7002**. Индекс слота определяется порядком следования объекта конфигурации соответствующего слота в массиве **slots**.

3.4 Примеры конфигурационных файлов

На рисунке 5 представлен пример конфигурационного файла для подключения одного АЦП без использования специальной конфигурации слотов.

Рисунок 5 – Конфигурация с одним АЦП без конкретизации слотов

В данном примере ПО при старте подключиться к АЦП с серийным номером 25005, сконфигурирует его порты на работу в режиме «АС» и создаст один слот по умолчанию. В слоте, созданном по умолчанию, количество каналов будет соответствовать количеству портов АЦП, к которому осуществилось подключение. При этом, порядок следования каналов будет соответствовать порядку следования портов в АЦП. HTTP-сервер будет прослушивать порт 7000/TCP.

На рисунке 6 представлена конфигурация ПО с настройкой генератора.

Рисунок 6 – Подключение к АЦП с генератором

Данный пример аналогичен предыдущему, отличием является настройка генератора в соответствии с таблицей 5. Генератор включается в режиме гармонического сигнала с частотой 100 Гц и амплитудой 1000 мВ.

В примере 7 представлен пример конфигурации в которой настраивается вибро-контроллер в соответствии с таблицей 4.

```
{
  "fatalFail": false,
  "initTime": 2000,
  "mod_id": "Стриммер с 1 АЦП с вибро-контроллером",
  "httpPort": 7000,
  "units": [
    {
      "id": "25005",
      "portsConfig": [{ "mode": "AC" }, { "mode": "AC" }],
      "vibroConfig": {
        "valueType": "A_RMS",
        "restoreMode": "manual",
        "highPass": 10,
        "lowPass": 20000,
        "relay1Lim": 5,
        "relay2Lim": 10,
        "relay3Lim": 15,
        "relay1Default": "NC",
        "relay2Default": "NO",
        "relay3Default": "NC",
        "relayOkDefault": 3,
        "overTime": 5000,
        "startPause": true,
        "rangeValuePeak": 1,
        "rangeValueRMS": 2,
        "rangeValueP2P": 3,
        "rangeValueGAP": 4,
        "rangeValueDIS": 1,
        "filter50": false,
        "medianaValue": 1
      }
    }
  ],
  "slots": []
}
```

Рисунок 7 – Пример конфигурации с использованием вибро-контроллера

Рисунок 8 – Конфигурация с двумя АЦП и двумя слотами

В представленном на рисунке 8 конфигурационном файле определены два устройства: 25025 и 25023. Для АЦП 25025 оба порта настроены в режиме AC. Для АЦП 25023 оба порта настроены в режиме IEPE. Определено два слота. Первый слот транслирует 4 канала, первые два канала связаны с портами АЦП 25025, вторые два канала связаны с портами АЦП 25023. Имена каналов выбраны таким образом, чтобы отражать связь каналов и портов между собой. Первый слот транслирует свои данные на порт 7001/TCP, второй слот - 7002/TCP.

4. Отказоустойчивый режим работы

Отдельно следует остановиться на алгоритме работы ПО в отказоустойчивом режиме. Основным параметром, определяющим работу ПО в отказоустойчивом режиме является параметр **fatalFail**, если он установлен в **true**, то ПО прекращает работу при потери связи с любой из АЦП (мастер, ведомый или одиночный – не имеет значения), т.е. работа ведётся не в отказоустойчивом режиме. Закрываются все TCP соединения, завершает работу HTTP-сервер.

В случае установки параметра **fatalFail** в значение **false,** ПО работает в отказоустойчивом режиме по следующим алгоритмам:

1) Потеря связи с ведомым устройством не отражается на работе клиентских программ, вместо показаний, снимаемых с ведомого устройства, в канал начинает транслироваться значение, указанное в параметре **defaultValue**;
2) Потеря связи с мастером приводит к остановке стека, разрываются все TCP-соединения для того, чтобы проинформировать клиентское ПО о том, что ему нужно будет выполнить повторное подключение. Далее, в зависимости от значения параметра **fixedMaster** ПО принимает решение о том назначить ли нового мастера (**fixedMaster** = **false**) и переинициализировать стек, либо остаться в состоянии ожидания появления потерянного мастера (**fixedMaster** = **true**). После переинициализации стека, TCP-серверы слотов будут доступны для подключений. В обоих случаях HTTP-сервер остается функционирующим и внешнее приложение может получить информацию о том, в каком состоянии находится ПО;
3) Потеря всех устройств стека приводит к тому, что ПО закрывает все TCP-соединения и переходит в режим ожидания восстановления связи с устройствами (хотя бы с одним). HTTP-сервер остается функционирующим и внешнее приложение может получить информацию о том, в каком состоянии находится ПО.
4) При восстановлении связи с потерянным ранее устройством, разрываются все TCP-соединения для того, чтобы проинформировать клиентское ПО, о том, что ему нужно будет выполнить повторное подключение, выполняется переинициализация стека. После переинициализации стека, TCP-серверы слотов будут доступны для подключений. HTTP-сервер остается функционирующим и внешнее приложение может получить информацию о том, в каком состоянии находится ПО;
5) В случае, если при старте за время, указанное в параметре **initTime**, не были найдены все АЦП, то вместо сигналов от отсутствующих АЦП будет транслироваться значение **defaultValue**.

5. Проверка программы

В процессе работы ПО осуществляет вывод в консоль служебных сообщений и сообщений об ошибках. При нормальной работе в консоль отражается информация о полученной конфигурации, либо о работе в режиме «по умолчанию», параметры АЦП с которым установлена связь (серийный номер, частота дискретизации, разрядность, количество входов и т.п.), информация о поступивших и разорванных TCP соединениях. Пример консольного вывода на рисунке 9.

```
16:31:29 [STREAMER][UNIT][25001][0] set channel mode [ IEPE ]
16:31:29 [STREAMER][UNIT][25001][1] set channel mode [ IEPE ]
16:31:29 [STREAMER][STACK] all units are ready
16:31:29 [STREAMER][STACK] starting [ CLUSTER ] mode
16:31:29 [STREAMER][STACK] master is [ 25007 ]
16:31:29 [STREAMER][STACK] state changed to [ RUNNING ]
16:31:29 [STREAMER][dataServer] init
16:31:29 [ADC][25001] register writen [ 311 ]
16:31:29 [ADC][25007] register writen [ 321 ]
16:31:29 [ADC][25001] register writen [ 312 ]
16:31:29 [ADC][25007] register writen [ 322 ]
16:31:29 [ADC][25001] register writen [ 202 ]
16:31:29 [ADC][25001] register writen [ 203 ]
16:31:29 [ADC][WRAPPERS::str_slave][25001] ADC started
16:31:29 [ADC][25007] register writen [ 202 ]
16:31:29 [ADC][25007] register writen [ 203 ]
16:31:29 [ADC][WRAPPERS::str_master][25007] ADC started
16:31:29 [ADC][25007][0] channel's state changed [ UNCONTROLLED ]
16:31:29 [ADC][25007][1] channel's state changed [ UNCONTROLLED ]
16:31:29 [ADC][25001][0] channel's state changed [ UNCONTROLLED ]
16:31:29 [ADC][25001][1] channel's state changed [ UNCONTROLLED ]
```

Рисунок 9 – Пример консольного вывода

Для проверки корректной работы ПО следует использовать приложение GTL. Для выполнения этой процедуры необходимо в менеджере устройств GTL выполнить добавление устройства типа ADCStream, как это показано на рисунке 10, с указанием в поле ID строки вида: 127.0.0.1:7000::0, где 127.0.0.1 – IP-адрес хоста на котором запущено ПО, 7000 – порт httpPort запущенного экземпляра ПО «Стриммер» и 0 – индекс слота, к которому выполняется подключение.

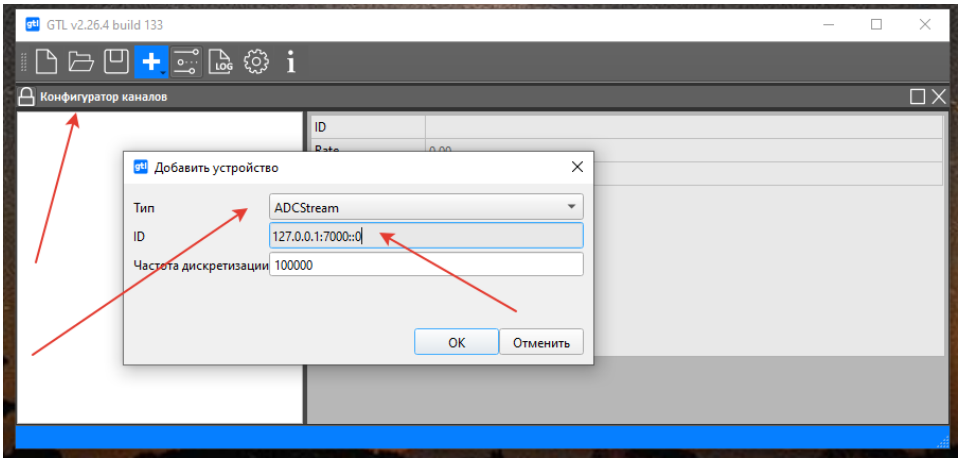

Рисунок 10 – Добавление устройства ADCStream в приложении GTL

В рассматриваемом примере конфигурации создано три слота с разным составом транслируемых входов АЦП, одним экземпляром приложения GTL можно осуществить подключение ко всем трем слотам, как это показано на рисунке 11.

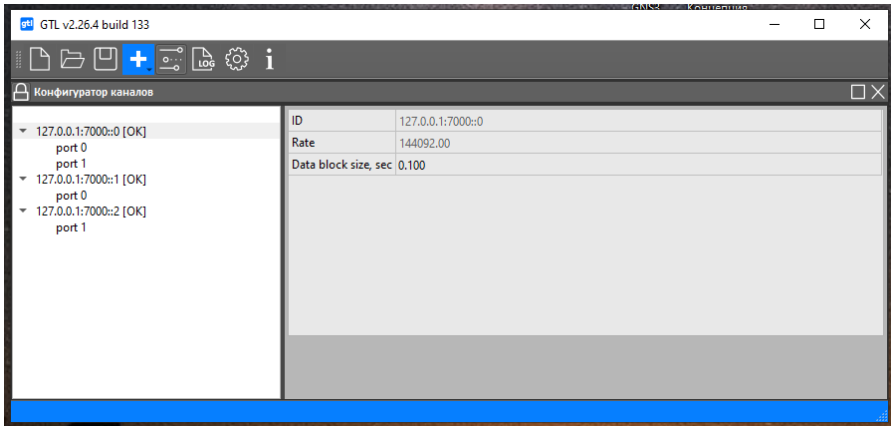

Рисунок 11 – Конфигуратор каналов GTL

Кроме этого, проверку корректности конфигурирования ПО можно осуществить с помощью WEB-браузера выполнив запрос http://127.0.0.1:7000/streamerConfig, где **127.0.0.1** – адрес хоста на котором запущено ПО и **7000** – порт, указанный в конфигурации ПО в параметре **httpPort**. Пример вывода конфигурации в браузере представлен на рисунке 12.


```
{
    "channelsCount": 4,
    "defaultValue": 0,
    "fatalFail": false,
    "fixedMaster": false,
    "initTime": 2000,
    "mod_id": "Стриммер с 2-мя АЦП и 2-мя слотами",
    "publisherURL": "http://172.16.205.33:3001/publish",
    "rate": 144092,
    "slots": [
        {
            "channels": [
                {
                    "name": "АЦП 1 [порт 0]",
                    "port": 0
                },
                {
                    "name": "АЦП 1 [порт 1]",
                    "port": 1
                },
                {
                    "name": "АЦП 2 [порт 0]",
                    "port": 2
                },
                {
                    "name": "АЦП 2 [порт 1]",
                    "port": 3
                }
            ],
            "dataPort": 7001
        },
        {
            "channels": [
                {
                    "name": "АЦП 2 [порт 1]",
                    "port": 3
                }
            ],
            "dataPort": 7002
        }
    ],
    "status": "RUNNING"
}
```


Рисунок 12 – Пример вывода конфигурации в браузере

6. Запуск Docker -контейнера

Скачать Docker-контейнер ПО «Стриммер» можно по сслыке:

https://downloads.gtlab.pro/files/streamer/Docker, как показано на рисунке 12.

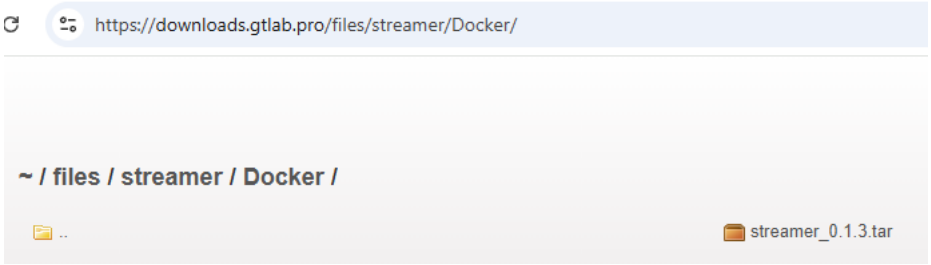

Рисунок 12 – Директория для скачивания Docker-контейнера ПО «Стриммер»

После скачивания контейнера следует выполнить команду: **docker load < streamer_0.1.3.tar**, как показано на рисунке 13.

```
root@stand5:/home/webmaster# docker load < streamer_0.1.3.tar
185e04da9d94: Loading layer [==================================================>]  123.8MB/123.8MB
fe59926ba509: Loading layer [==================================================>]  39.15MB/39.15MB
fdb52belafd0: Loading layer [==================================================>]  441.2MB/441.2MB
Loaded image: streamer:0.1.3-6.1
```

Рисунок 13 - Результат выполнения команды docker load

Для проверки корректности загрузки докер контейнера можно выполнить команду **docker images**, как показано на рисунке 14.

```
root@stand5:/home/webmaster# docker images
REPOSITORY   TAG          IMAGE ID       CREATED        SIZE
streamer     0.1.3-6.1    14766ale4b60   14 hours ago   597MB
```

Рисунок 14 – Список контейнеров

Для запуска Docker-контейнера следует использовать следующую конфигурацию docker-compose, представленную на рисунке 15.

```
version: "3"
services:
    streamer:
        container_name: streamer
        image: streamer:0.1.3-6.1
        restart: always
        volumes:
            - /home/streamer/:/home/streamer/
        environment:
            - CONFIG_FILENAME=/home/streamer/sl.json
        ports:
            - "7000:7000"
            - "7001:7001"
            - "7002:7002"
            - "7003:7003"
```

Рисунок 15 – Конфигурация docker-compose

При составлении пускового файла docker-compose.yml важно указать все TCP-порты которые должны будут использоваться при работе с ПО, а также путь к конфигурационному файлу через переменную окружения CONFIG_FILENAME.

7. Интеграция с разрабатываемыми приложениями

При разработке приложений, совместимых с ПО «Стриммер», следует руководствоваться информацией, представленной в данном разделе.

Первое действие которое должно выполнить клиентское приложение - это чтение конфигурации путём выполнение HTTP-GET –запроса на адрес:

http://127.0.0.1:7000/streamerConfig

где **127.0.0.1** – это IP–адрес хоста, на котором запущено ПО «Стриммер».

В ответ на данный запрос будет получена текущая конфигурация в формате JSON, как показано на рисунке 16.


```
{
    "channelsCount": 4,
    "defaultValue": 0,
    "fatalFail": false,
    "fixedMaster": false,
    "initTime": 2000,
    "mod_id": "Стриммер с 2-мя АЦП и 2-мя слотами",
    "publisherURL": "http://172.16.205.33:3001/publish",
    "rate": 144092,
    "slots": [
        {
            "channels": [
                {
                    "name": "АЦП 1 [порт 0]",
                    "port": 0
                },
                {
                    "name": "АЦП 1 [порт 1]",
                    "port": 1
                },
                {
                    "name": "АЦП 2 [порт 0]",
                    "port": 2
                },
                {
                    "name": "АЦП 2 [порт 1]",
                    "port": 3
                }
            ],
            "dataPort": 7001
        },
        {
            "channels": [
                {
                    "name": "АЦП 2 [порт 1]",
                    "port": 3
                }
            ],
            "dataPort": 7002
        }
    ],
    "status": "RUNNING"
}
```


Рисунок 16 – Пример конфигурации

Структура представленной на рисунке конфигурации схожа со структурой конфигурационного файла и не требует каких-то дополнительных уточнений. Из конфигурации видно, что работа ведётся со стеком АЦП, суммарное количество портов в котором – **4** (параметр - **channelsCount**). Частота дискретизации АЦП – **144092** Гц (параметр **rate**). Конфигурацией предусмотрено два слота. Первый слот

содержит 4 канала, данные которых транслируются через TCP-сервер, прослушивающий порт 7001/TCP. Второй слот содержит один канал, данные которого транслируются через TCP-сервер, прослушивающий порт 7002/TCP.

Для получения данных с нужного слота (с требуемым набором портов АЦП) необходимо подключиться к соответствующему TCP-серверу, используя номер порта из параметра **dataPort**. После подключения клиент сразу начнёт получать поток данных, которые передаются слотом без дополнительных запросов. Данные передаются в формате float32 (по 4 байта на каждое значение). Весь поток можно разделить на сэмплы: каждый сэмпл содержит данные с каналов слота, идут они подряд в том же порядке, в котором каналы указаны в конфигурации слота.

На рисунке 17 представлена иллюстрация потоков данных для примера трёх слотов. В первом слоте два канала, во втором и третьем – по одному.

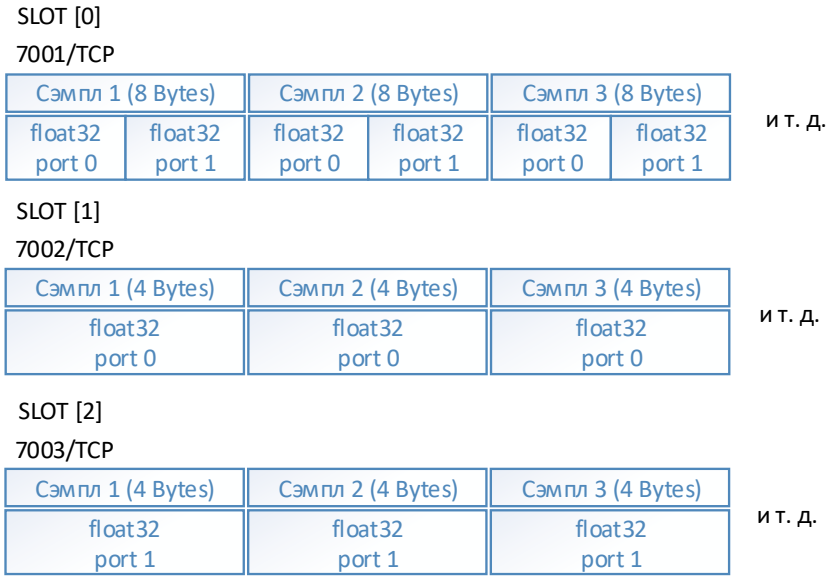

Рисунок 17 – Поток данных слотов

Имея данные о том, сколько каналов передаётся через тот или иной слот, а также частоту дискретизации АЦП, клиентское приложение может восстановить исходный сигнал по получаемым данным.

Данные отправляются порциями с периодичностью 50 мс. В этом случае, объем отправляемых данных будет зависеть от количества каналов, транслируемых слотом, и частоты дискретизации АЦП.

Количество выбираемых из сокета байт с данными следует подбирать таким образом, чтобы оно составляло число кратное размеру одного сэмпла. Размер одного сэмпла составляет 4 * количество каналов в слоте.

Пример кода на C++ с использованием библиотеки Qt, представлен на рисунке 18.

```
qint64 __bytesCount = __socket->bytesAvailable();

if (__bytesCount >= _count_ai * DATA_TYPE_SIZE &&
    _count_ai > 0) {

    int __sampleSize = _count_ai * DATA_TYPE_SIZE;
    int __samples = __bytesCount / __sampleSize;

    _buffer_read.resize(__samples * _count_ai);

    QByteArray __data = __socket->read(__samples * __sampleSize);

    for (int __sampleNumber = 0; __sampleNumber < __samples; __sampleNumber++) {

        for (int __channelNumber = 0; __channelNumber < _count_ai; __channelNumber++) {
            customTypes::bytes2float __value;

            int __offset = __sampleSize * __sampleNumber + __channelNumber * DATA_TYPE_SIZE;

            __value.bytes.HH = __data.data()[__offset];
            __value.bytes.HL = __data.data()[__offset + 1];
            __value.bytes.LH = __data.data()[__offset + 2];
            __value.bytes.LL = __data.data()[__offset + 3];

            _buffer_read[__sampleNumber * _count_ai + __channelNumber] = __value.floatValue;
        }

    }
}
```

Рисунок 18 – Фрагмент кода обработки потока данных

Комментарии к представленному фрагменту кода:

1) Переменная **_buffer_read** определена как std::vector<float>;
2) Описание типа **bytes2float** представлено на рисунке 19;

```
namespace customTypes {

    struct couple { ...};

    union bytes2word { ...};

    struct twinCouple {
        uint8_t LL;
        uint8_t LH;
        uint8_t HL;
        uint8_t HH;
    };

    union bytes2long { ...};

    union bytes2float {
        twinCouple bytes;
        float floatValue;
    };

    union byte2bits {          ...};

}
```

Рисунок 19 – Описание типа byte2float

3) Переменная **__socket** является экземпляром класса **QTcpSocket.**

7.1 Методы мониторинга состояний

Встроенный HTTP-сервер, кроме метода запроса конфигурации (**streamerConfig**) предоставляет 3 метода мониторинга состояний ПО «Стриммер»:

- **/slotState** – состояние слота, предоставляет данные по состоянию каналов слота, используется с параметром index (индекс слота);
- **/stackState** – состояние стека, предоставляет данные о всех АЦП и о состоянии их портов;
- **/connectionsState** – предоставляет данные о всех активных клиентских подключениях к слотам.

Данные методы можно использовать для организации мониторинга состояния ПО «Стриммер» внешним приложением.

7.1.1 Метод slotState

Данный метод применяется для доступа к актуальной информации о состоянии слота, используется с параметром index.

Запрос вида: http://127.0.0.1/slotState?index=0, выдаст информацию о состоянию слота с индексом 0. На рисунке 19 представлено два результата выполнения метода:

а) оба АЦП на связи и передают данные;

б) потеряна связь с АЦП 25001.

а) все АЦП на связи


```
{
    "channels": [
        {
            "adcPort": 0,
            "isOnline": true,
            "portState": "UNCONTROLLED",
            "sn": 25007
        },
        {
            "adcPort": 1,
            "isOnline": true,
            "portState": "UNCONTROLLED",
            "sn": 25007
        },
        {
            "adcPort": 0,
            "isOnline": true,
            "portState": "UNCONTROLLED",
            "sn": 25001
        },
        {
            "adcPort": 1,
            "isOnline": true,
            "portState": "UNCONTROLLED",
            "sn": 25001
        }
    ],
    "slotStatus": "RUNNING"
}
```


б) один АЦП отключен


```
{
    "channels": [
        {
            "adcPort": 0,
            "isOnline": true,
            "portState": "UNCONTROLLED",
            "sn": 25007
        },
        {
            "adcPort": 1,
            "isOnline": true,
            "portState": "UNCONTROLLED",
            "sn": 25007
        },
        {
            "adcPort": 0,
            "isOnline": false,
            "sn": 25001
        },
        {
            "adcPort": 1,
            "isOnline": false,
            "sn": 25001
        }
    ],
    "slotStatus": "RUNNING_DEGRADED"
}
```


Рисунок 19 – Результат выполнения метода slotState

Таблица 8 – Значение ключей объекта slotState

Ключ	**Тип значения**	**Диапазон**	**Комментарий**
slotStatus	String	RUNNING	Работает в полной конфигурации
		RUNNING_DEGRADED	Работает в неполной конфигурации
		FAILED	Связь со всеми АЦП потеряна
		UNDEFINED	Запрашиваемый слот не найден
channels	[]	[]	Массив объектов состояний каналов

Таблица 9 – Значение ключей объекта состояния канала

Ключ	**Тип значения**	**Диапазон**	**Комментарий**
adcPort	Int	Int	Номер порта АЦП
sn	Int	Int	Серийный номер АЦП
portState	String	IEPE_DISABLED	IEPE отключен
		IEPE_OK	IEPE включен, датчик подключен
		IEPE_SHORT	IEPE включен, короткое замыкание
		IEPE_OPEN	IEPE включен, обрыв
		UNCONTROLLED	Нет аппаратного механизма контроля
		TACHO	Тахо сигнал без поддержки IEPE
inOnline	Bool	True	АЦП на связи
		False	АЦП не на связи

7.1.2 Метод stackState

Данный метод применяется для доступа к актуальной информации об аппаратном состоянии стека АЦП.

Запрос вида: http://127.0.0.1/stackState, выдаст информацию об аппаратном состоянии стека АЦП. На рисунке 20 представлено два результата выполнения метода:

а) оба АЦП на связи и передают данные;

б) потеряна связь с АЦП 25001.

а) все АЦП на связи


```
{
    "stackStatus": "RUNNING",
    "unitStates": [
        {
            "address": "192.168.205.74",
            "isMaster": true,
            "isOnline": true,
            "ports": [
                "UNCONTROLLED",
                "UNCONTROLLED"
            ],
            "sn": "25007",
            "unitStatus": "RUNNING"
        },
        {
            "address": "192.168.205.75",
            "isOnline": true,
            "ports": [
                "UNCONTROLLED",
                "UNCONTROLLED"
            ],
            "sn": "25001",
            "unitStatus": "RUNNING"
        }
    ]
}
```


б) один АЦП отключен


```
{
    "stackStatus": "RUNNING_DEGRADED",
    "unitStates": [
        {
            "address": "192.168.205.74",
            "isMaster": true,
            "isOnline": true,
            "ports": [
                "UNCONTROLLED",
                "UNCONTROLLED"
            ],
            "sn": "25007",
            "unitStatus": "RUNNING"
        },
        {
            "address": "192.168.205.75",
            "isOnline": false,
            "sn": "25001",
            "unitStatus": "DISCONNECTED"
        }
    ]
}
```


Рисунок 20 – Результат выполнения метода stackState

Таблица 10 – Значение ключей объекта stackState

Ключ	**Тип значения**	**Диапазон**	**Комментарий**
stackStatus	String	CREATED	Создан
		STARTING	Запускается
		RUNNING	Работает в полной конфигурации
		RUNNING_DEGRADED	Работает в неполной конфигурации
		FAILED_MASTER	Связь с мастером потеряна
		FAILED_ALL	Связь со всеми АЦП потеряна
		FAILED_COMPATIBILITY	Ошибка совместимости АЦП
		FAILED_SINGLE	Связь с АЦП потеряна (одиночный режим)
unitStates	[]	[]	Массив объектов состояний АЦП

Таблица 11 – Значение ключей объекта состояния АЦП

Ключ	**Тип значения**	**Диапазон**	**Комментарий**
address	String	String	IP-адрес АЦП
sn	Int	Int	Серийный номер АЦП
isMaster	Bool	True	Работает в режиме ведущего (ключ не используется у ведомых)
unitStatus	String	DISCOVER	Поиск
		CONNECTING	Подключается
		CONNECTED	Подключено
		CONFIGURATION_READING	Чтение конфигурации
		WAITING_COMMAND	Ожидает команды
		ADC_START_SEND	Команда «старт» отправлена
		ADC_STOP_SEND	Команда «стоп» отправлена
		INIT_FAILED	Ошибка инициализации
		DISCONNECTED	Отключено
		RUNNING	Работает
ports	[]	[]	Массив состояний портов *Значения аналогичны параметру **portState**, представленному в таблице 9*
inOnline	Bool	True	АЦП на связи
		False	АЦП не на связи

7.1.3 Метод connectionsState

Данный метод применяется для доступа к актуальной информации об клиентских подключениях в ПО «Стриммер».

Запрос вида: http://127.0.0.1/connectionsState, выдаст информацию о всех активных клиентских соединениях к каждому из слотов. На рисунке 21 представлен результат выполнения данного метода.

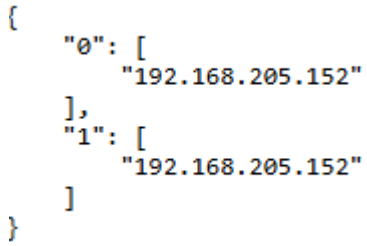

Рисунок 21 – Пример выполнения connectionsState

Из рисунка видно, что у каждого из двух слотов есть по одному активному клиентскому подключению с IP-адреса 192.168.205.152.

7.2 Использование интерфейса PUBLISHER

Второй способ мониторинга состояния ПО «Стриммер» - использование интерфейса PUBLISHER. При активации интерфейса ПО будет выполнять периодические и асинхронные HTTP-POST запросы на указанный URL. В периодических запросах содержится информация о текущем состоянии ПО: объекты **stackState** и **connectionsState**. В асинхронных запросах передаются сообщения о событиях (отключение и подключение АЦП, изменения состояния портов АЦП, подключение и отключение клиентов и т.п.). Подробнее с перечнем событий, поддерживаемых ПО «Стриммер», можно ознакомиться в [2].

Настройки интерфейса представлены в таблице 1, к ним относятся ключи: **publisherURL**, **publisherIntervalMs**, **stateRouteKey** и **eventsRouteKey**. Подробнее о формате HTTP-POST запросов, применяемых в интерфейсе PUBLISHER можно ознакомиться в [1].

На рисунке 22 представлен пример периодической отправки состояния, а на рисунке 23 пример сообщения о событии.

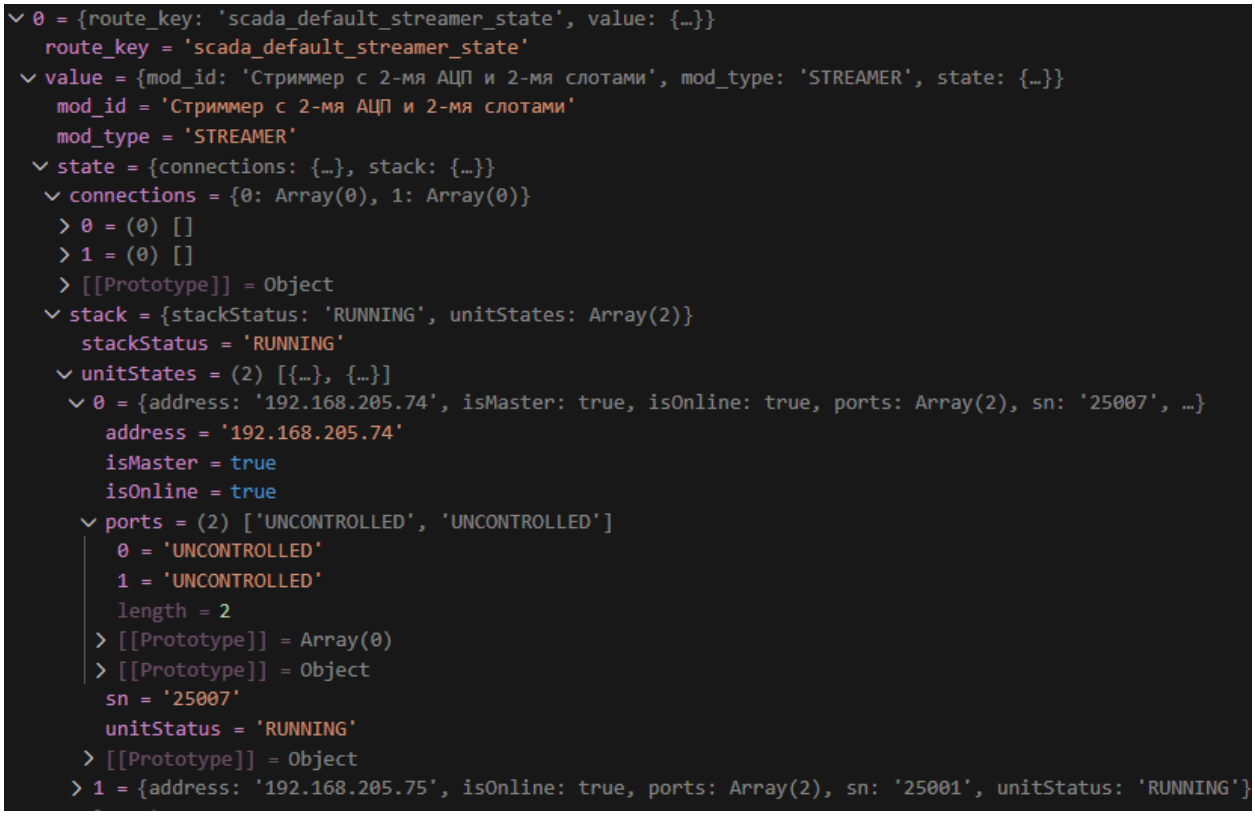

Рисунок 22 – Пример объекта состояния

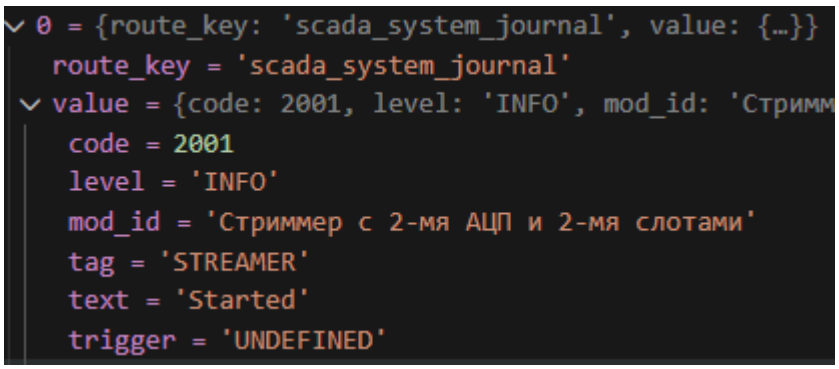

Рисунок 23 – Пример объекта события

Источники, использованные при разработке

1) ООО «ГТЛАБ-Диагностика». GTLD-SCADA-PUBLISHER. Простой интерфейс публикации данных: Руководство системного программиста. Саров, 2025.
2) ООО «ГТЛАБ-Диагностика». GTLD-SCADA. Перечень зарезервированных кодов событий: Руководство системного программиста. Саров, 2025.