МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет **ФИЗИЧЕСКИЙ**

Кафедра **АВТОМАТИЗАЦИИ ФИЗИКО-ТЕХНИЧЕСКИХ ИССЛЕДОВАНИЙ**

Направление подготовки **03.03.02 ФИЗИКА**

Образовательная программа: **БАКАЛАВРИАТ**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Ключниковой Анны Александровны

(Фамилия, Имя, Отчество автора)

Тема работы «Разработка средства верификации систем управления установками электронного охлаждения»

«К защите допущена»	**Научный руководитель**
Заведующий кафедрой	к.ф.-м.н.
к.т.н.	Зав. лаб. 5-2 ИЯФ СО РАН
… Лысаков К. Ф …/……..	… Рева В. Б …/……..
(фамилия И., О.) / (подпись, МП)	(фамилия И., О.) / (подпись, МП)
«……»……20…г.	«……»……20…г.

Дата защиты: «……»……20…г.

Новосибирск, 2019

Оглавление

1. Введение

Электронное охлаждение – эффективный современный метод увеличения фазовой плотности ионных пучков в ускорителях заряженных частиц. Основу метода составляет теплообмен между холодными электронами и горячими ионами на некотором общем участке траектории. В настоящее время данный метод интенсивно развивается в проекте ускорительно комплекса NICA, (ОИЯИ, г. Дубна), где заложено две системы электронного охлаждения на 50 кэВ и 2.5 МэВ (ИЯФ СО РАН). Обе установки представляют собой сложный физико-технический комплекс состоящих из большого числа различных подсистем, которыми необходимо управлять и контролировать. Для этого в настоящее время создается комплекс программного обеспечения, включающий в себя управление магнитным полем установки, магнитными корректорами, высоковольтным терминалом, системой защит и др. Одним из ключевых факторов в написании надежно работающего физического комплекса является верификация программного обеспечения. Создание симулятора установки является одним из способов такой проверки. Это особенно актуально, когда использование установок и их создание происходят в разных городах. В этом случае необходим инструмент, позволяющий проверять корректность работы удалённой установки при внесении каких-либо модификаций в программный комплекс, ведь затраты на устранение ошибок как правило намного больше, чем проработка их заранее, до внедрения нового оборудования или программного обеспечения.

Рассмотрим установку на 50 кэВ для ускорительного комплекса NICA. В настоящее время она установлена и проведен ее технический запуск. На рисунке 1 приведена ее фотография в процессе пусконаладочных работ.

Рисунок 1. Фотография установки электронного охлаждения в процессе пуско-наладочных работ.

С точки зрения управления она состоит из следующих подсистем показанных на рисунке 2. Система высоковольтного терминала ”SGF” для управления источниками питания электронной пушки и коллектора, источником высокого напряжения 0-60 кВ и источником питания электростатических пластин, служащих для поворотов электронов на тороидальных участках магнитного поля, источники питания корректирующих магнитов ”MPS”, система контроля мощных источников питания (IST) для создания продольного магнитного поля, система блокировки и сигнализации ”UBS”, система управления генератором тактирующих импульсов CGTI, система измерений положения пучка с помощью пикап-электродов и отдельного блока

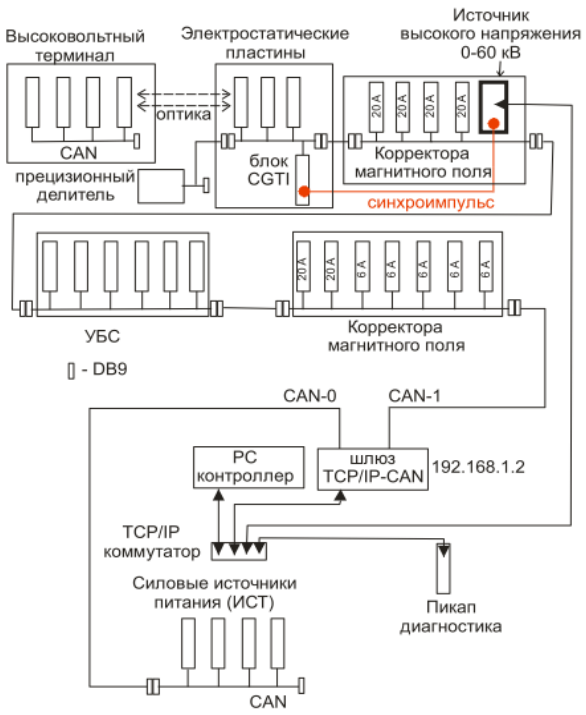

Рисунок 2. Схема автоматизации установки электронного охлаждения

прецизионного делителя для измерения энергии электронов с относительной точностью до 10^{-4}. В качестве низкоуровневой внутренней шины для обмена информацией в процессе работы между различными измерительными и управляющими блоками, входящими в состав установки, используется линия типа CANbus (Controller area network – локальная сеть микроконтроллеров). При работе СЭО в составе бустера планируется осуществлять два режима работы. Первый, “стационарный”, когда все значения напряжений и токов в корректорах -постоянны во времени и охлаждение происходит на одной фиксированной энергии. Второй, “бустерный”, когда осуществляется одновременное охлаждение на двух уровнях энергии (одно в районе инжекции, другое в области E-cooling) в течении одного цикла. Предполагаемая диаграмма работы СЭО в “бустерном” режиме показана на рис.3.

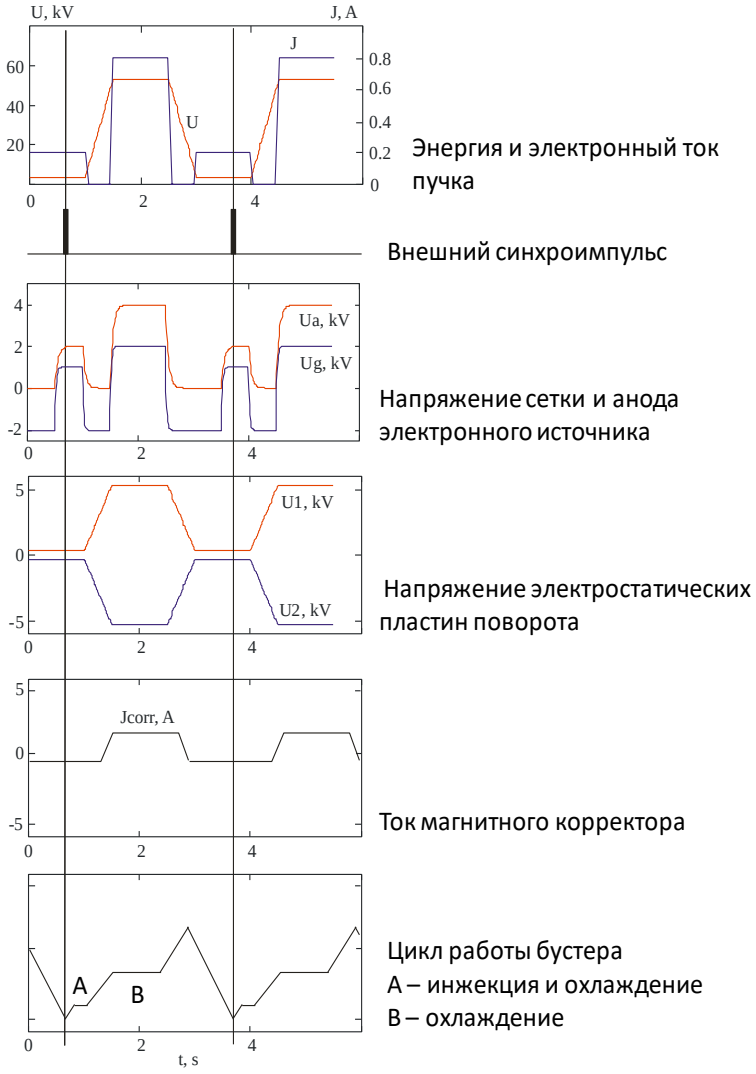

Рисунок 3. Предполагаемая диаграмма работы СЭО в бустерном режиме

На время перестройки высокого напряжения, электростатических пластин и корректоров – ток пучка выключается с помощью подачи нулевого напряжения на анод и отрицательного на сетку. Поэтому временная форма перестройки высокого напряжения, корректоров и напряжения электростатических пластин не особо важна, главное, чтобы все значения успевали прийти в заданную точку ко времени возникновения электронного тока в пушке. Как видно из рисунка 2 связь между подсистемами: высоковольтный терминал, электростатические пластины и корректора магнитного поля осуществляется по единой линии связи CAN, что дает возможность синхронизовать работу различных источников. Для этого соответствующие таблицы должны быть загружены в управляющие модули, после чего они запускаются от единой команды от блока CGTI с использованием линии связи CAN.

В данный момент описанная установка работает на локальном программном обеспечении, созданном в ИЯФ СО РАН. Но проект NICA использует распределенную систему управления Tango в качестве официальной системы управления. Так как данная установка используется в этом проекте, то для интеграции её в единую систему необходимо осуществить переход от локального программного обеспечения на Tango – то есть оставить действующую функциональность, сделав возможным использовать протокол управления оборудованием на основе TANGO. Система верификации может помочь осуществить этот переход с наименьшими потерями, так как она позволит проверять работу установки дистанционно - на эмуляторах блоков. Tango Controls – свободно распространяющаяся система управления любыми видами аппаратных и программных систем, которая поддерживает языки программирования C++, Java и Python. Она позволяет отображать на мониторе огромное количество данных в режиме реального времени, обрабатывать и сохранять в базах данных информацию для последующей обработки, оповещения в реальном времени и в целом зарекомендовала себя как надежное средство.

Данная работа посвящена разработке средства верификации программного обеспечения систем управления установками электронного охлаждения.

С целью создания средства, позволяющего моделировать системы электронного охлаждения и проверять корректность работы программного обеспечения для данных систем.

Работа состоит из введения, 5 глав, заключения, списка литературы и приложения. В 1 главе сделан обзор существующих методов верификации программного обеспечения, обзор протокола CAN, библиотеки ZeroMQ для обеспечения взаимодействия виртуальных устройств, а также обзор распределенных систем управления. В конце главы сформулированы требования к системе и поставлены конкретные задачи. Во 2 главе сделана постановка задачи данной работы. В 3 главе описана система верификации – ее составные части, схема взаимодействия частей, а также руководство по использованию. В 4 главе приведены примеры взаимодействия разработанной системы верификации с программным обеспечением, действующем на реальной установке, а также скрипты, запускающие промоделированные устройства для некоторых подсистем установки электронного охлаждения ИЯФ СО РАН. В 5 главе предложены расширения разработанной системы верификации.

2. Обзор литературы

2.1 Подходы к виртуальному тестированию

В современном мире способы верификации делят на [1, с. 138]:

- эмпирические - те, которые используют экспертизу – они не могут быть автоматизированы, т.к. выполняются экспертами. Зато такая верификация может проводиться на любом этапе разработки и зависит лишь от компетенций специалистов.
- формальные, использующие математический аппарат верификации программного обеспечения. Тут проверяется не программный код, а математическая модель системы, её соответствие спецификации. Минусом такого метода является отсутствие в некоторых случаях возможности создать адекватную формальную модель программы. Но с другой стороны, есть возможность автоматизации такого тестирования, возможность увидеть исходный код в виде схем, логических выражений и оценить таким образом соответствие спецификации.
- формальные, проверяющие работу программы с помощью запуска.

Методы верификации также классифицируют:

- Статический – тестирование без исполнения программы. Это дедуктивный анализ и метод проверки модели. В дедуктивном методе задаются пред и пост условия, а затем (как правило вручную) делаются выводы и просчеты. Метод проверки модели включает в себя построение математической модели на основе программного кода и затем её анализ на предмет существующих ограничений и условий. Такие методы не зависят от среды или компилятора, позволяют обнаружить ошибки из-за копирования частей программы, но они мало эффективны при ошибках, связанных с утечками памяти. То есть, статические методы полезны на этапе разработки программного обеспечения, но мало эффективны на завершающих этапах.

- Динамический – мониторинг и тестирование, непосредственно во время исполнения программы. Один из методов – мониторинг. С помощью него осуществляется оценка работы программы, проверка корректности, регистрация каких-то данных, действий. Мониторинг может использовать специальные инструменты. Как правило используются какие-либо известные сценарии поведения. Динамическое исследование позволяет найти множество стандартных ошибок, а также дает найти некоторые виды вирусов, появляющихся при запуске программы. Также осуществляется поиск ошибок работы с оперативной памятью, что особенно актуально для многопоточных приложений. Динамические методы включают в себя разработку тестовой модели системы и динамическую генерацию тестов.
- Синтетические методы – некая комбинация статических и динамических методом, компенсирующая недостатки каждого из них. Примерами таких методов служат тестирование на основе моделей и мониторинг формальных свойств [2]

В рамках данной классификации созданную систему верификации можно отнести к динамической системе, т.к. корректность оборудования проверяется при запуске и непосредственной его работе.

Существуют инструменты для автоматизации тестирования. Среди инструментов управления информацией о тестах известны следующие продукты: TestManager от IBM/Rational и TestDirector от HP/Mercury. Существуют также инструменты генерации тестовых данных, например, генератор сложных тестовых данных на основе грамматик языка SynTESK от ИСП РАН. Выделяют также инструменты тестирования пользовательского интерфейса, например, Windows GUI, GUI библиотек KDE или Gnome для Linux, WebUI [2, с. 87]

Можно заметить, что методы верификации сильно зависят от программируемой системы, а что также, нет единого инструмента, подходящего для каждой области и каждой задачи. Поэтому, для систем электронного охлаждения и

других систем, использующих блоки CANbus, требуется независимое средство верификации программного обеспечения, учитывающее особенности используемого в них протокола CAN и самих аппаратных блоков.

2.2 Протокол CANbus

В установках электронного охлаждения активно используется протокол CANbus. Выбор данного протокола обусловлен некоторыми преимуществами перед другими распространенными протоколами, такими, как RS-232, RS-485, Ethernet. Во время изучения этих протоколов были выявлены преимущества протокола CAN, которые описаны далее.

Протокол RS-485 позволяет устройствам обмениваться сообщениями в полудуплексном режиме, как и стандарт CAN. Но так как стандарт RS-485 работает только на физическом уровне, то он не определяет структуру сообщений, не определяет схему взаимодействия, такие механизмы, как разрешение коллизий, предоставляются разработчику и т.д. Единственное правило арбитража, возможное в данном стандарте – Master/Slave, что не всегда удобно. Из-за улучшенному механизма разрешения ошибок, CAN является более отказоустойчивым стандартом и считается надежным и устойчивым, а также оставляя свободным интерфейс UART, дает возможность использовать его для отладки. [3]

Стандарт RS-232 асинхронного интерфейса UART применяется для связи компьютеров с модемами и другими периферийными устройствами. Этот стандарт также определен только на физическом уровне, предполагает, как асинхронный, так и синхронный режим передачи данных. В настоящее время интерфейс UART вытесняется интерфейсом USB [4].

В отличие от CAN, Ethernet не обеспечивает надежную доставку самостоятельно – для этого необходим такой протокол, как TCP/IP. Согласно [5] CAN со скоростью 1 Mbps быстрее передает кадры, чем стандартный Ethernet-TCP/IP со скоростью передачи 10 Mbps. Так, 8 байт полезной информации при

100% загрузке шины передается со скоростью 8700 FPS для CAN, а для стандартного 10 Mbps Ethernet при загрузке шины на 20% – скорость передачи кадров составляет около 2900 FPS.

Так как требуется промоделировать работу устройств с интерфейсом CANbus, то необходимо было изучить особенности данного протокола. Изначально данный протокол был придуман для автомобильной электроники, но он оказался очень удобным для использования в других областях промышленности и науки. Шина CAN осуществляет последовательную передачу пакетов с разрешением коллизий и является широковещательной, то есть все устройства получают все сообщения. Таким образом на одной линии может быть несколько устройств (до 64). Стандарт CAN активно используется в ИЯФ СО РАН с 2000г. В протоколе CAN сообщения состоят из следующих частей:

1. Поле идентификатора (11 или 29 бит)
2. Поле длины данных (1 байт)
3. Поле данных (до 8 байт)

Идентификатор определяет приоритетность сообщения, а также то, какому устройству (или устройствам) данное сообщение адресовано. Таким образом, любое устройство на линии получает все сообщения, распознает идентификатор сообщения и на его основе понимает, адресовано ли ему данное сообщение. В установках электронного охлаждения используется множество CAN устройств, поэтому для изучения и моделирования выбраны наиболее распространены.

- CEDIO_A [6] - многопортовый регистр ввода/вывода с интерфейсом CANbus для ввода и вывода дискретной информации. Оно состоит из следующих компонент:
    - 16-разрядный выходной регистр;
    - 16-разрядный входной ТТЛ регистр;
    - генераторы сообщений по внешнему событию;
    - CANBUS интерфейс, по которому осуществляется связь устройства с управляющей ЭВМ;

- встроенный микропроцессор.

- CEAD20 [7] – устройство, которое осуществляет контроль напряжений источников питания в системах управления ускорительных комплексов, работу в системах термоконтроля, а также используется в качестве универсального АЦП широкого применения. Оно состоит из:
  - 20/40 канальный АЦП
  - 8 канальный выходной регистр с гальванически изолированными выходами;
  - 8 канальный входной регистр с гальванически изолированными входами;
  - CANBUS интерфейс, по которому осуществляется связь устройства с управляющей ЭВМ;
  - встроенный микропроцессор.
- CEAC124 [8] - устройство контролирует напряжение источников питания, работу в системах термоконтроля, а также как универсальное АЦП широкого применения. Состав устройства:
  - 12 канальный АЦП
  - 4 канальный выходной регистр с гальванически изолированными выходами;
  - 4 канальный входной регистр с гальванически изолированными входами;
  - CANBUS интерфейс, по которому осуществляется связь устройства с управляющей ЭВМ;
  - встроенный микропроцессор.

2.3 Библиотека ZeroMQ

Для взаимодействия виртуальных устройств необходимо было спроектировать способ обмена сообщениями. Библиотека обмена сообщениями ZeroMQ, позиционирующаяся, как достаточно простой программный интерфейс для со-

здания собственной системы обмена сообщениями, отвечает всем требованиям для моделирования системы, а именно – асинхронный обмен сообщениями между различными устройствами и процессами, компромисс между сложностью реализации взаимодействия и высокой производительностью, поддержка нескольких языков программирования. Библиотека помогает разработчику практически не заботиться о буферизации данных, обслуживании очередей, восстановлении и установлении соединений. Минусом является то, что библиотека не отвечает за сохранность сообщений и не гарантирует доставку, за этим должен следить разработчик, также нет явного управления очередями и при переполнении новые сообщения отбрасываются, но найденные недостатки приемлемы в рамках поставленной задачи [9]. ZeroMQ предоставляет:

- Кроссплатформенность
- Использование различных современных языков программирования, например, C++, Java, Python, Go, R, Ruby.
- Пересылку сообщений между процессами, позволяет использовать IPC, TCP, TIPC, многоадресную рассылку
- Паттерны типа pub-sub, push-pull, router-dealer, request-reply
- Асинхронность
- Свободное распространение

2.4 Системы управления

Для осуществления перехода на новое программное обеспечение необходимо было изучить существующие системы управления.

Tango Controls не зависит от аппаратной части, что дает возможность использовать собственные драйверы для соединения с ней, а так как эта система разрабатывается уже много лет, то многие драйверы уже существуют в свободном каталоге и их можно использовать для собственных нужд. Поддерживает

Tango позволяет управлять большими и маленькими системами, при этом каждая система имеет централизованную базу данных (MariaDB/MySQL. База

данных хранит конфигурацию устройства для запуска девайс-сервера и действует как сервер имен, сохраняя динамические сетевые адреса. Коммуникационный протокол определяет, как все компоненты системы будут взаимодействовать друг с другом. Tango использует CORBA (Common Object Request Broker Architecture) и ZeroMQ. Детали этого протокола скрыта от разработчика и используются встроенными высокоуровневыми средствами. Tango Controls предоставляет различные средства [10]:

- Jive для конфигурации компонент системы и загрузки статической базы данных Tango
- AtkPanel помогает генерировать панели контроля
- Pogo – генератор девайс-классов и девайс-серверов
- Taurus – Python библиотека для создания интерфейса приложения.

Кроме TANGO, одной из самых распространенных и проработанных систем является EPICS (Experimental Physics and Industrial Control System). Она используется во всем мире для создания распределенных систем управления научными установками, такими как ускорители частиц, телескопы. EPICS – это:

- архитектура для построения масштабируемых систем управления;
- сборник кода и документации, включающий программный инструментарий;
- сотрудничество крупных научных лабораторий и промышленности.

В работе [11] было проведено сравнение систем TANGO и EPICS при управлении базовой системой. Оказалось, что:

1. В TANGO и EPICS используются разные взгляды на вещи. EPICS управляет технологической переменной, которая является элементарной информацией, поступающей от контролируемого оборудования. В TANGO основным элементом является «устройство». Внутри устройства вы найдете команды, атрибуты и состояния. В EPICS нет такого понятия устройства, в котором:

- Вы можете группировать информацию (атрибуты)
- У вас также есть команды (для выполнения действий с управляемым оборудованием)
- У вас есть состояние устройства, позволяющее вам построить конечный автомат.

Поэтому, используя TANGO, вы можете создавать иерархию информации.

2. В TANGO система представляется на более абстрактном уровне, чем в EPICS, где у вас есть набор распределенных объектов, предоставляющих сервисы, команды и состояния, и вы можете определить иерархию между ними.
3. По мнению автора статьи [11], для использования Tango нужно иметь опыт программирования, хотя это не критичное требование. EPICS же не требует каких-либо навыков программирования для базовых операций и имеет больше «промышленного контроля», чем «информатики». Поэтому, если есть команда программистов, то использование TANGO предпочтительнее, но если системы будут строиться специалистами по оборудованию (инженерами), то удобнее использовать EPICS.

Таким образом обе системы достаточно удобны и выбор одной из них – совокупность таких факторов, как особенности системы, особенности команды разработчиков. Tango - это продукт, ориентированный на компьютерную науку и предназначенный для использования в качестве полного пакета для систем управления. Так как использование TANGO – выбор учредителей проекта NICA – то именно эта система и должна быть использована для управления установкой.

3. Постановка задачи

Целью данной работы является разработка средства, позволяющего моделировать системы электронного охлаждения и проверять корректность работы программного обеспечения для данных систем, для последующего перехода на новое программное обеспечение удалённой установки электронного охлаждения.

Программное управление каждой подсистемой системы электронного охлаждения состоит из сервера, работающего с оборудованием и обеспечивающего трансляцию TCP запросов в запросы CAN, и клиента, работающего только с сервером. Сервера полностью отвечают за логику работы с оборудованием, а клиенты обеспечивают логику работы оператора. При этом если, сервер на каждую подсистему может быть только один, то количество клиентов не ограниченно, как по количеству экземпляров, так и их типов. Для того чтобы промоделировать любую из подсистем существующей установки, нужно:

1. Модифицировать работу шлюза TCP/IP-CAN таким образом, чтобы сервер, взаимодействующий с блоками управления, не заметил подмены реальных устройств на виртуальные. В таком случае, модифицированный шлюз можно будет использовать для любой системы, взаимодействующей посредством CAN устройств и TCP/IP приложений.
2. Промоделировать устройства, входящие в состав конкретной подсистемы
3. Иметь возможность расширять список промоделированных устройств для создания других систем, использующих интерфейс CANbus.

Это, автоматически, позволяет проверять как работу серверной, так и клиентской части программного обеспечения.

4. Описание библиотеки

За основу библиотеки взята библиотека CanGW 1.14 [12]. В сервере can4lgw, осуществляющем трансляцию TCP/IP сообщений в CAN, модифицированы запросы и запись параметров контроллера, его перезапуск, а также вместо ioctl команд на открытие/закрытие портов, запись и чтение, реализованы запросы к виртуальной шине CAN с помощью ZeroMQ сокетов. Стоит заметить, что Шлюз TCP/IP←→CAN работает под операционной системой Linux разрядности 32 бит – это обусловлено в первую очередь тем, что реальный шлюз – основа данного – использует именно эту систему. Аргументы в пользу данного выбора приведены в работе [13] – основные это:

1. Распространенность.
2. Поддержка POSIX.
3. Большой набор доступных программных компонент.

Шлюз написан на языке C. Остальные части библиотеки реализованы на языке Python, так как для использования и расширения видов устройств этот язык удобнее и быстрее – не требуется компиляция, распространен во многих научных институтах. Также, для функционирования системы необходима библиотека ZeroMQ для языков C и Python. Схема взаимодействия частей системы верификации представлена на рисунке 4.

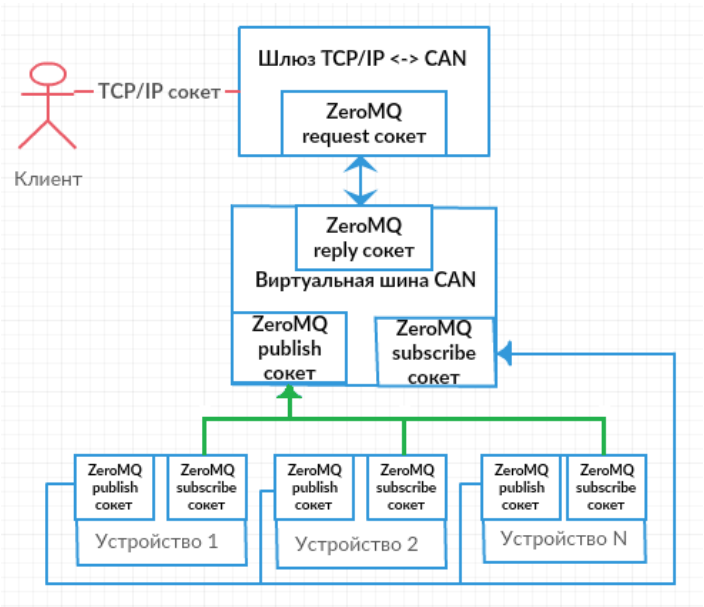

Рисунок 4. Схема взаимодействия системы верификации

Основные части библиотеки:

- Шлюз TCP/IP←→CAN (написан на языке C)

Транслирует TCP/IP запросы от клиентов в CAN пакеты и в обратную сторону. Запросы к устройствам отправляются в виртуальную шину CAN через ZeroMQ request/reply сокет. Данный паттерн позволяет удобно взаимодействовать с устройствами в формате master-slave, который требуется для управления установкой.

- Виртуальная шина CAN (написан на языке Python)

Это отдельный сервер, который слушает запросы от шлюза и отправляет сообщения виртуальным устройствам через ZeroMQ publish сокет. ZeroMQ subscribe сокет служит для приема сообщений от виртуальных устройств. Эти сообщения сохраняются во внутренний кольцевой FIFO буфер для отправки в шлюз при соответствующем запросе от него. Выбор паттерна publish/subscribe для данных сокетов обусловлен тем, что шина CAN является широковещательной, поэтому её работа и была проэмулирована таким образом.

- Виртуальные устройства (написаны на языке Python)

Это отдельные классы, моделирующие поведение реальных CAN устройств. Они взаимодействую с виртуальной шиной CAN через два ZeroMQ сокета – publish и subscribe. Они наследуются от шаблона виртуальных устройств, который описан в главе 4.1.

- Executor для старта работы виртуальных устройств (написан на языке Python)

Данный класс позволяет удобно запускать виртуальные устройства в работу, особенно, когда их несколько в одной подсистеме электронного охлаждения. Примеры приведены в приложении.

Клиентские программы подключаются по протоколу TCP/IP к шлюзу. При запросе открытия порта от клиентского приложения, шлюз связывает ZeroMQ сокет с заданным адресом. Через этот сокет осуществляется прием и передача сообщений в виртуальную шину CAN. Этот сервер, в свою очередь, создает

два сокета: через один (ZeroMQ publish сокет) отправляются сообщения виртуальным устройствам, а через второй (ZeroMQ subscribe сокет) принимаются сообщения от виртуальных устройств.

Для запуска системы необходимо запустить скрипт bash с именем StartVirtualSystem в операционной системе Linux разрядности 32 бита. Данный скрипт запускает виртуальный шлюз TCP/IP←→CAN и два сервера виртуальных CAN шин. В качестве аргументов можно передать адреса сокетов для виртуальных шин, а также адреса сокетов для взаимодействия с виртуальными устройствами в формате:

1. «хост:порт» виртуальной CAN шины 0 (по умолчанию 127.0.0.1:5555)
2. «хост:порт» для отправки сообщений из виртуальной CAN шины 0 виртуальным устройствам (по умолчанию 127.0.0.1:5556)
3. «хост:порт» для получения сообщений из виртуальной CAN шины а 0 виртуальным устройствам (по умолчанию 127.0.0.1:5557)
4. «хост:порт» виртуальной CAN шины 1 (по умолчанию 127.0.0.1:6665)
5. «хост:порт» для отправки сообщений из виртуальной CAN шины 1 виртуальным устройствам (по умолчанию 127.0.0.1:6666)
6. «хост:порт» для получения сообщений из виртуальной CAN шины 1 виртуальным устройствам (по умолчанию 127.0.0.1:6667)

Передать можно от одного до шести аргументов, соблюдая порядок.

Виртуальные устройства можно запустить как на ОС Linux, так и ОС Windows. Пример взаимодействия с системой верификации приведен в главе 5 данной работы.

Шлюз TCP/IP←→CAN является модифицированной версией сервера can4lgw. Подробное описание шлюза TCP/IP←→CAN представлено в [12]. API шлюза осталось тем же, модифицированы все функции, кроме:

- void dump_server_stat (cangw_server_conn_t *srv)
- void sigusr1(int sig)

- int kbhit (void)
- void help_and_exit (const char *prog)

Обращение к реальным устройствам заменено взаимодействием с виртуальным CAN портом по ZeroMQ сокету в режиме запрос-ответ, путем пересылки массива байт (см. рис. 4). Виртуальная CAN шина осуществляет разбор входящего запроса. Возможные запросы:

- прочитать из CAN шины входящие сообщения:

Виртуальная CAN шина отправляет все накопившиеся во внутреннем буфере сообщения в шлюз единым массивом байт.

- отправить запрос виртуальным CAN устройствам на линии:

Виртуальная CAN шина отправляет пришедший от шлюза запрос виртуальным устройствам через ZeroMQ сокет в режиме издатель-подписчик. В данном режиме запрос приходит всем виртуальным устройствам, подключенным к этому сокет, а задача устройства определить по содержимому запроса, ему оно адресовано или нет и как его обработать. Ответные посылки виртуальные устройства отправляют на виртуальную CAN шину через второй ZeroMQ сокет в режиме издатель-подписчик и сохраняются в кольцевой буфер.

4.1 Шаблон виртуальных устройств

Для создания виртуального устройства создан родительский класс-шаблон (на языке Python). На рис. 5 представлена диаграмма классов виртуальных устройств с основными функциями некоторых классов. Подробное описание промоделированных виртуальных устройств приведено в главах 4.3 - 4.7

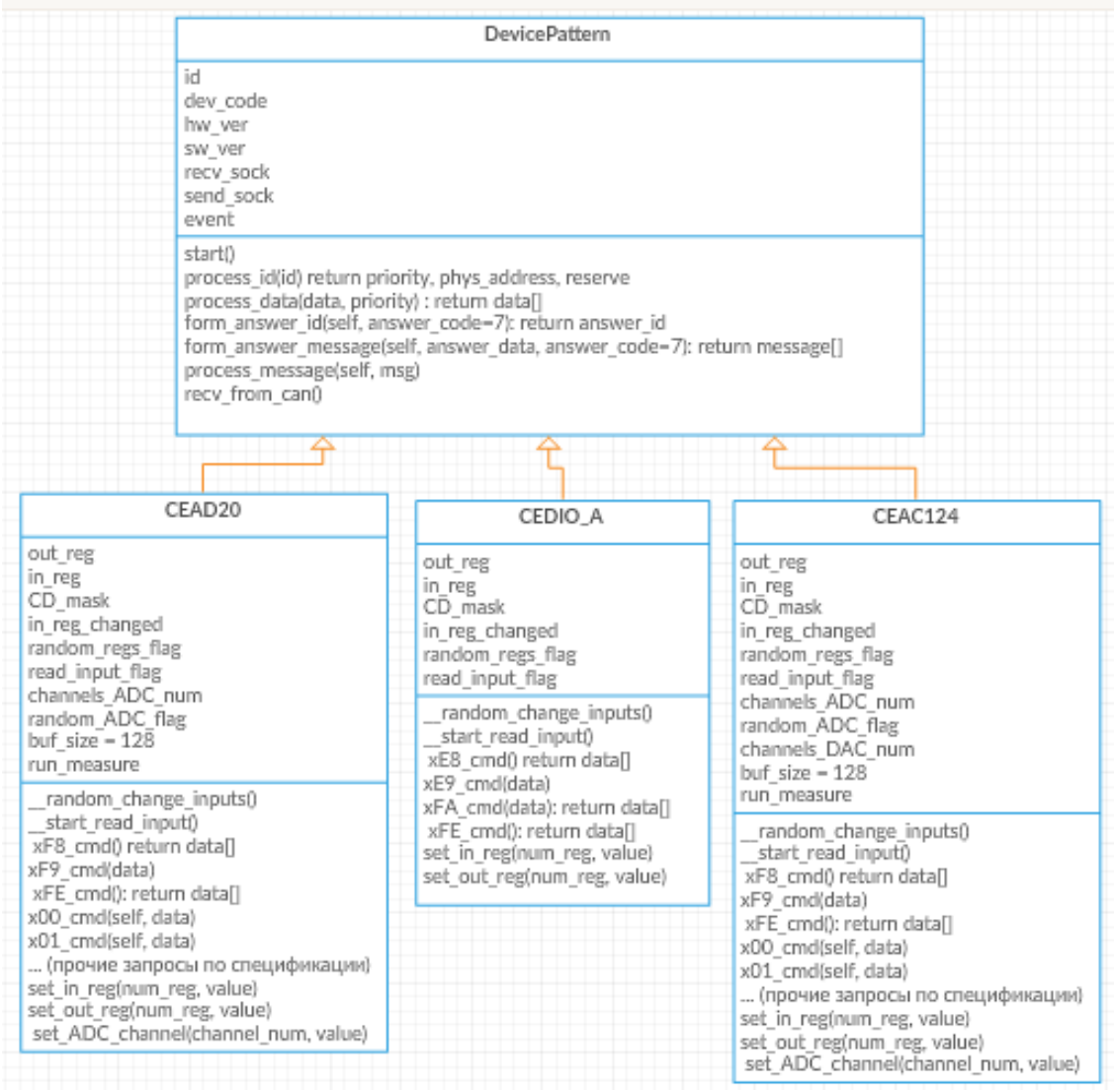

Рисунок 5. Диаграмма классов виртуальных устройств

Для создания нового устройства необходимы следующие действия:

- Унаследовать новое виртуальное устройство от шаблона
- Добавить необходимые поля, такие как виртуальные регистры, виртуальные каналы и т.д.
- Добавить симуляцию необходимых процессов (например, изменение значений регистров, каналов)

- В функцию process_data(self, data, priority) добавить обработку всех необходимых запросов таким образом, чтобы функция либо вернула ответные данные, длиной до 8 байт включительно, либо не возвращала ничего - если запрос не требует ответа. Номер команды находится в 0 элементе массива data.
- Для работы с несколькими нитями (используются для моделирования работы устройств – такой, как изменение регистров, каналов АЦП случайным образом, либо по какому-то алгоритму) все устройства имеют параметр event – событие threading.event() - общее на все устройства в скрипте. По умолчанию его значение None и требуется либо передать его в конструктор, либо установить с помощью функции set_event(event)

Интерфейс конструктора содержит следующие параметры:

- send_addr – адрес сокета для отправки посылок в виртуальный порт (стандартно tcp://192.168.43.105:5556)
- recv_addr – адрес сокета для получения посылок из виртуального порта (стандартно tcp://192.168.43.105:5557)
- event – событие threading.event() - общее на все устройства для работы с нитями.
- id – адрес устройства (стандартно 0)
- dev_code – код устройства (стандартно 0)
- hw_ver – версия аппаратной части (стандартно 1)
- sw_ver – версия программного обеспечения (стандартно 1)

Также, все устройства имеют возможность получать команды на установления значений регистров и каналов АЦП от пользователя из консоли непосредственно во время работы виртуального устройства. Для этого нужно выставить соответствующему флагу значение true. Формат команд, которые для выставления значений нужно написать в консоли и название флага приведены в описании каждого устройства в данной главе.

Таким образом, наследуясь от данного шаблона, в новом виртуальном устройстве необходимо переопределить функцию process_data(args), добавив новые команды и добавить необходимые поля и функции, отвечающие за поведение устройства, соответствующие элементам и поведению реального устройства, а обработка запросов от внешних программ и формирование ответных посылок уже реализованы в родительском классе.

4.2 «Активатор» виртуальных устройств

Для запуска виртуального устройства, используется класс Executor. Для инициализации необходим ZeroMQ poller для опроса сокетов всех виртуальных устройств, которые будут запущены, и список экземпляров этих устройств. На рис. 6 представлена агрегация класса Executor и классов виртуальных устройств.

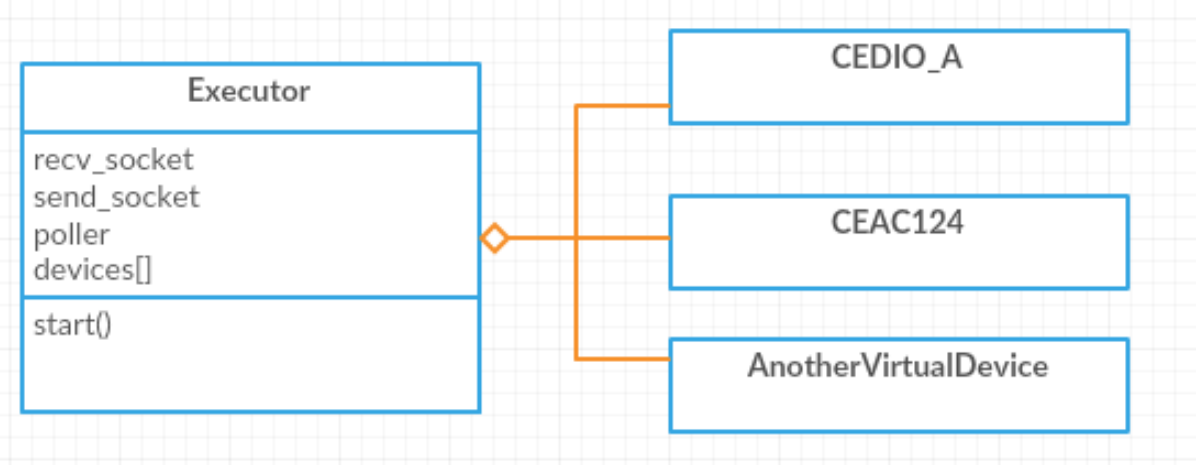

Рисунок 6. Диаграмма отношения классов Executor и классов виртуальных устройств

Функция start(self) создает нить, в которой poller слушает сокеты переданных устройств и при наличии входящих сообщений, Executor запускает их обработку. На рис. 7 представлена диаграмма управления виртуальным устройством с помощью Executor.

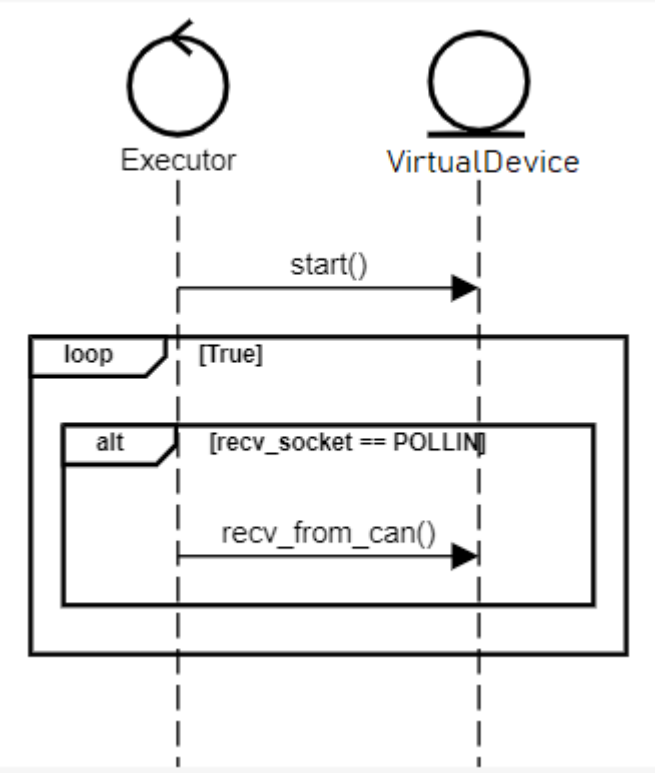

Рисунок 7. Диаграмма управления виртуальными устройствами с помощью Executor

Рассмотрим пример кода:

```
event = threading.Event()
devices = [CEAC124.CEAC124(event=event)]
poller = zmq.Poller()
my_executor = Executor(poller, devices, event)
my_executor.start()
```

В первой строке создается событие для многопоточности. Во второй строке создается список устройств для запуска. В данном случае это одно устройство CEAC124. Третьей строкой инициализируется поллер, который будет контролировать, когда в сокет виртуального устройства приходят сообщения, чтобы executor запустил обработку этих сообщений. В четвертой строке инициализируется executor и в пятой, с помощью функции start, executor запускает устройство CEAC124.

4.3 Виртуальное устройство CEDIO_A

Устройство отвечает на команды согласно спецификации [6]. Также интерфейс включает функции:

- установить значение входного регистра set_in_reg(self, num_reg, value), с параметрами num_reg – номер регистра и value - значение – 0 или 1
- установить значение выходного регистра set_out_reg(self, num_reg, value), с параметрами num_reg – номер регистра и value - значение – 0 или 1

При создании устройства, кроме параметров базового класса, в конструктор можно передать следующие параметры:

- read_input_flag – флаг, устанавливающий ожидание считывания следующих команд пользователя из консоли (стандартно True):
  - Установить значение входного регистра в формате: Device_id In Num_of_register Register_value
  - Установить значение входного регистра в формате: Device_id Out Num_of_register Register_value
- random_regs_flag – флаг, устанавливающий случайное изменение значений регистров с течением времени (стандартно False)

Стандартное значение dev_code = 28.

Например, данный кусок кода:

```
device = CEDIO_A(event=event, id=1,
random_regs_flag=True)
```

создает экземпляр класса CEDIO_A с адресом 1 и случайно изменяющимися значениями регистров. Остальные параметры стандартные.

4.4 Виртуальное устройство CEDIO_D

Устройство отвечает на команды согласно спецификации. Также интерфейс включает функции:

- установить значение входного регистра set_in_reg(self, num_reg, value), с параметрами num_reg – номер регистра и value - значение – 0 или 1
- установить значение выходного регистра set_out_reg(self, num_reg, value), с параметрами num_reg – номер регистра и value - значение – 0 или 1

При создании устройства, кроме параметров базового класса, в конструктор можно передать следующие параметры:

- read_input_flag – флаг, устанавливающий ожидание считывания следующих команд пользователя из консоли (стандартно True):
    - Установить значение входного регистра в формате: Device_id In Num_of_register Register_value
    - Установить значение входного регистра в формате: Device_id Out Num_of_register Register_value
- random_regs_flag – флаг, устанавливающий случайное изменение значений регистров с течением времени (стандартно False)

Стандартное значение dev_code = 42.

Например, данный кусок кода:

```
device = CEDIO_D(event=event, id=0x04,
send_addr="tcp://192.168.43.105:5556")
```

создает экземпляр класса CEDIO_D с адресом 4, которое отправляет сообщения в виртуальный порт в сокет с адресом 192.168.43.105:5556. Остальные параметры стандартные.

4.5 Виртуальное устройство CEAD20

Устройство отвечает на команды согласно спецификации [7]. Также интерфейс включает функции:

- установить значение входного регистра set_in_reg(self, num_reg, value), с параметрами num_reg – номер регистра и value - значение – 0 или 1
- установить значение выходного регистра set_out_reg(self, num_reg, value), с параметрами num_reg – номер регистра и value - значение – 0 или 1
- установить значение канала АЦП set_ADC_channel(self, channel_num, value), с параметрами channel_num – номер канала, value – значение канала

При создании устройства, кроме параметров базового класса, в конструктор можно передать следующие параметры:

- read_input_flag – флаг, устанавливающий ожидание считывания следующих команд пользователя из консоли (стандартно True):
    - Установить значение входного регистра в формате: Device_id In Num_of_register Register_value
    - Установить значение входного регистра в формате: Device_id Out Num_of_register Register_value
- random_regs_flag – флаг, устанавливающий случайное изменение значений регистров с течением времени (стандартно False)
- ADC_n – количество каналов АЦП (стандартно 20)
- in_reg – количество входных регистров (стандартно 4)
- out_reg – количество выходных регистров (стандартно 4)
- random_ADC_flag – флаг, устанавливающий случайное изменение значений каналов АЦП с течением времени (стандартно False)

Стандартное значение dev_code = 23.

Например, данный кусок кода:

```
device = CEAD20(event=event,id=0x1a, random_ADC_flag=True)
```

создает экземпляр класса CEAD20 с адресом 1a в шестнадцатеричной системе исчисления, которое случайным образом меняет значения каналов АЦП. Остальные параметры стандартные.

4.6 Виртуальное устройство CEAC124

Устройство отвечает на команды согласно спецификации [8]. Также интерфейс включает функции:

- установить входной регистр set_in_reg(self, num_reg, value), с параметрами num_reg – номер регистра и value - значение – 0 или 1
- установить выходной регистр set_out_reg(self, num_reg, value), с параметрами num_reg – номер регистра и value - значение – 0 или 1

- установить значение канала АЦП set_ADC_channel(self, channel_num, value), с параметрами channel_num – номер канала, value – значение канала

При создании устройства, кроме параметров базового класса, в конструктор можно передать следующие параметры:

- read_input_flag – флаг, устанавливающий ожидание считывания следующих команд пользователя из консоли (стандартно True):
    - Установить значение входного регистра в формате: Device_id In Num_of_register Register_value
    - Установить значение входного регистра в формате: Device_id Out Num_of_register Register_value
- random_regs_flag – флаг, устанавливающий случайное изменение значений регистров с течением времени (стандартно False)
- ADC_n – количество каналов АЦП (стандартно 12)
- in_reg – количество входных регистров (стандартно 4)
- out_reg – количество выходных регистров (стандартно 4)
- random_ADC_flag – флаг, устанавливающий случайное изменение значений каналов АЦП с течением времени (стандартно False)
- DAC_n – количество (стандартно 4)
- read_input_flag – флаг, устанавливающий ожидание считывания следующих команд пользователя из консоли (стандартно True):
    - Установить значение входного регистра в формате: Device_id In Num_of_register Register_value
    - Установить значение входного регистра в формате: Device_id Out Num_of_register Register_value
    - Установить значение канала АЦП в формате: Device_id ADC ADC_channel channel_value

Стандартное значение dev_code = 20.

Например, следующий кусок кода:

```
ceac = CEAC124.CEAC124(event=event)
```

создает экземпляр класса со стандартными параметрами.

4.7 Виртуальное устройство CEAC208

Устройство отвечает на команды согласно спецификации [14]. Также интерфейс включает функции:

- установить входной регистр set_in_reg(self, num_reg, value), с параметрами num_reg – номер регистра и value - значение – 0 или 1
- установить выходной регистр set_out_reg(self, num_reg, value), с параметрами num_reg – номер регистра и value - значение – 0 или 1
- установить значение канала АЦП set_ADC_channel(self, channel_num, value), с параметрами channel_num – номер канала, value – значение канала

При создании устройства, кроме параметров базового класса, в конструктор можно передать следующие параметры:

- read_input_flag – флаг, устанавливающий ожидание считывания следующих команд пользователя из консоли (стандартно True):
    - Установить значение входного регистра в формате: Device_id In Num_of_register Register_value
    - Установить значение входного регистра в формате: Device_id Out Num_of_register Register_value
- random_regs_flag – флаг, устанавливающий случайное изменение значений регистров с течением времени (стандартно False)
- ADC_n – количество каналов АЦП (стандартно 24)
- in_reg – количество входных регистров (стандартно 8)
- out_reg – количество выходных регистров (стандартно 8)
- random_ADC_flag – флаг, устанавливающий случайное изменение значений каналов АЦП с течением времени (стандартно False)
- DAC_n – количество (стандартно 8)

- read_input_flag – флаг, устанавливающий ожидание считывания следующих команд пользователя из консоли (стандартно True):
    - Установить значение входного регистра в формате:
      Device_id In Num_of_register Register_value
    - Установить значение входного регистра в формате:
      Device_id Out Num_of_register Register_value
    - Установить значение канала АЦП в формате:
      Device_id ADC ADC_channel channel_value

Стандартное значение dev_code = 4.

Например, с помощью следующего куска кода:

```
ceac = CEAC208(event=event, read_input_flag=False)
```

создается экземпляр класса, который не будет считывать команды пользователя из консоли и с остальными стандартными параметрами.

5. Примеры использования средства верификации

Примеры и тесты запускались на компьютере HP Laptop 14-cm0013ur с 4 ядерным процессором Ryzen 5 2500U и оперативной памятью 8Gb. Для использования системы запускается ОС Linux Ubuntu 16.04 LTS 32 бит на виртуальной машине Virtual Box версии 6.0.2 (Qt5.6.2). Далее в данной системе запускается скрипт StartVirtualSystem (см. главу 4 данной работы) без параметров (используются стандартные). Следующим шагом является запуск виртуальных блоков. В приведенных в этой главе примерах скрипты с использованием виртуальных блоков запускались в основной системе Windows 10, со стандартными адресами сокетов на получение и отправку сообщений в один из двух виртуальных портов.

5.1 Тестовый скрипт

Запустим тестовый скрипт test_script.py на python, запускающий виртуальное устройство CEDIO_A на основной операционной системе Windows со стандартными параметрами на нулевом виртуальном порту, случайным изменением значений регистров и адресом устройства равным 1. Теперь можно подключить TCP/IP клиент к виртуальному шлюзу. Для примера запускается CanGwMonitor (см. рис. 8), открывается нулевой порт по IP адресу виртуальной машины (кнопка open). Можно отправить широковещательный запрос FF и увидеть ответную посылку. Пример заполнения полей для нажатия кнопки send при отправке запроса приведен на рис. 8.

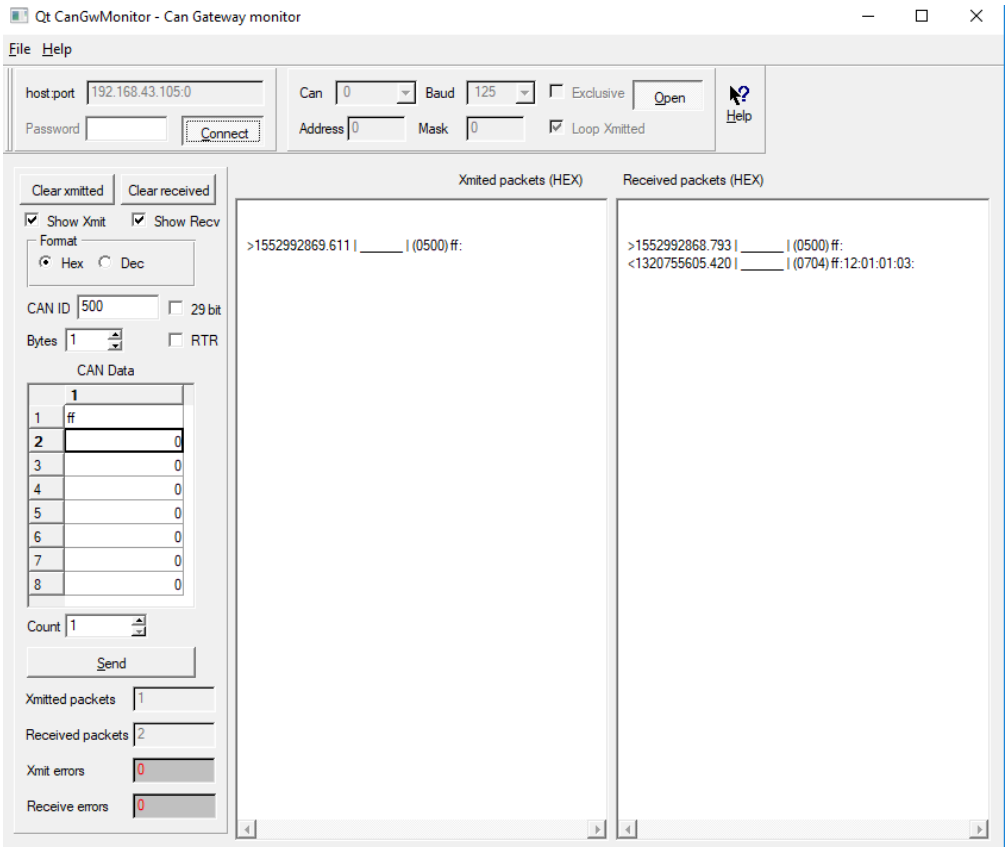

Рисунок 8. Пример отправки запроса через CanGwMonitor

5.2 Взаимодействие системы верификации с серверами SGF, UBS, MPS, IST

Были написаны скрипты sgf_script.py, ubs_script.py, mps_script.py, ist_script.py на языке Python (скрипты приведены в приложении), запускающие все необходимые виртуальные устройства для различных частей систем электронного охлаждения. Скрипты запускались на той же виртуальной машине с ОС Linux, а также на ОС Windows 10. Для тестов использовались реальные сервера и клиенты взятые с действующей установки электронного охлаждения (ОИЯИ, г. Дубна).

На рис. 9 виден результат взаимодействия трех виртуальных CEAC124 и двух CEDIO_D в скрипте sgf_script.py с сервером и клиентом SGF.

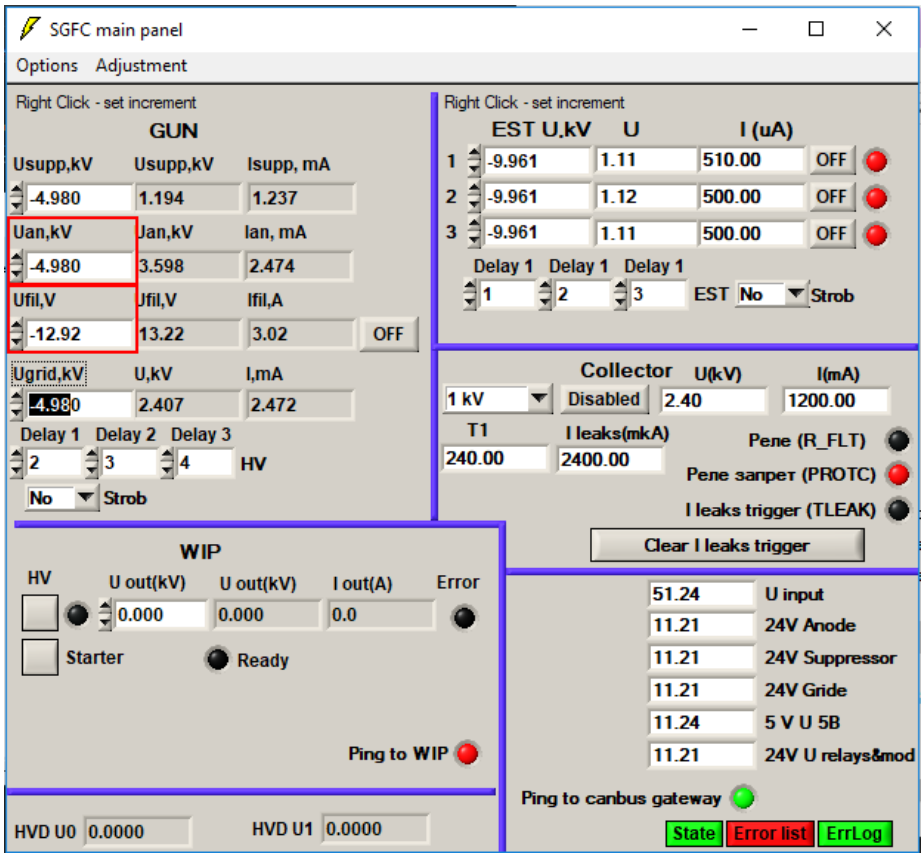

Рисунок 9. Взаимодействие с сервером и клиентом SGF

Для сравнения укажем скриншот взятого с запуска данной установки. На рис. 10 можно видеть показания подсистемы SGF при работе установки с энергией электронного пучка 17.5 кВ и током 150 мА.

Видно, что в отличие от клиента запущенного в режиме эмуляции, большая часть индикации выполнена в “зеленом” цвете. Это связано с тем, что данный клиент не только интерфейс взаимодействия системы и оператора, но также выполняет проверку данных полученных от устройств на техническую “корректность”.

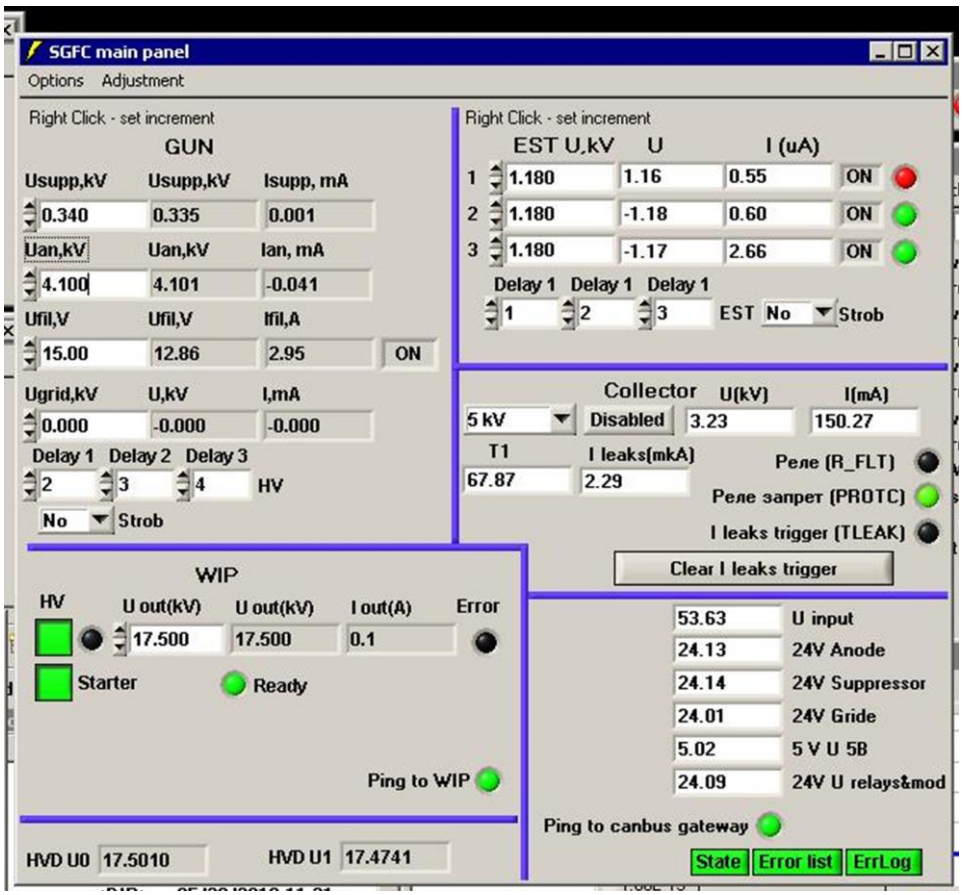

Рисунок 10. Скриншот работы сервера с реальной установкой

Следующий тест был проведен с помощью сервера и клиента UBS. На рис.11 виден результат взаимодействия шести виртуальных CEDIO_A и одного CEAD20 в скрипте ubs_script.py с сервером и клиентом UBS. На скриншоте Следующий тест был проведен с помощью сервера и клиента UBS. На рис.10 виден результат взаимодействия шести виртуальных CEDIO_A и одного CEAD20 в скрипте ubs_script.py с сервером и клиентом UBS. На скриншоте видно, что фиксируется изменение регистров виртуальных устройств CEDIO_A, а также каналов АЦП виртуального CEAD20.

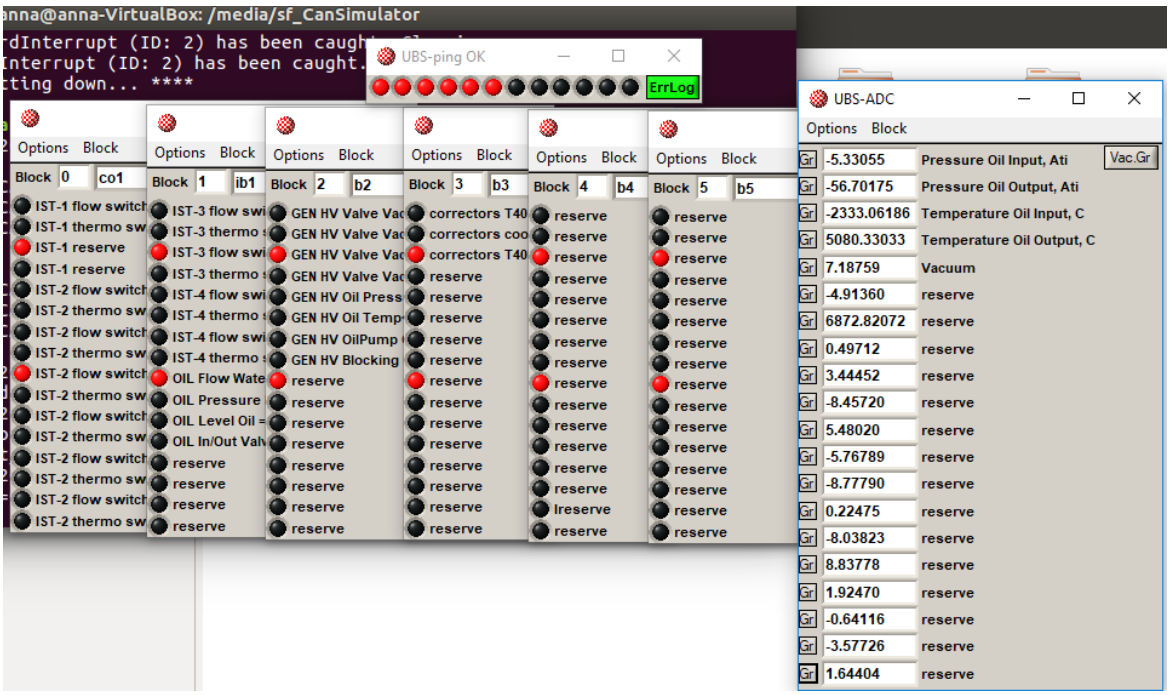

Рисунок 11. Взаимодействие с сервером и клиентом UBS

Далее было проведено взаимодействие виртуальной системы с сервером MPS, также использующимся с действующей установкой электронного охлаждения. На рис. 12 виден результат взаимодействия восьми виртуальных CEAC124 и восьми CAC208 в скрипте mps_script.py с сервером и клиентом MPS. Во время теста, кроме проверки команд сервера, из клиента вручную выставлялись значения ЦАПа. Видно, что значения выставлены, а красный цвет в некоторых ячейках, так же, как и в случае с остальными серверами, означает некорректную величину значения в канале с точки зрения реальных ожидаемых данных.

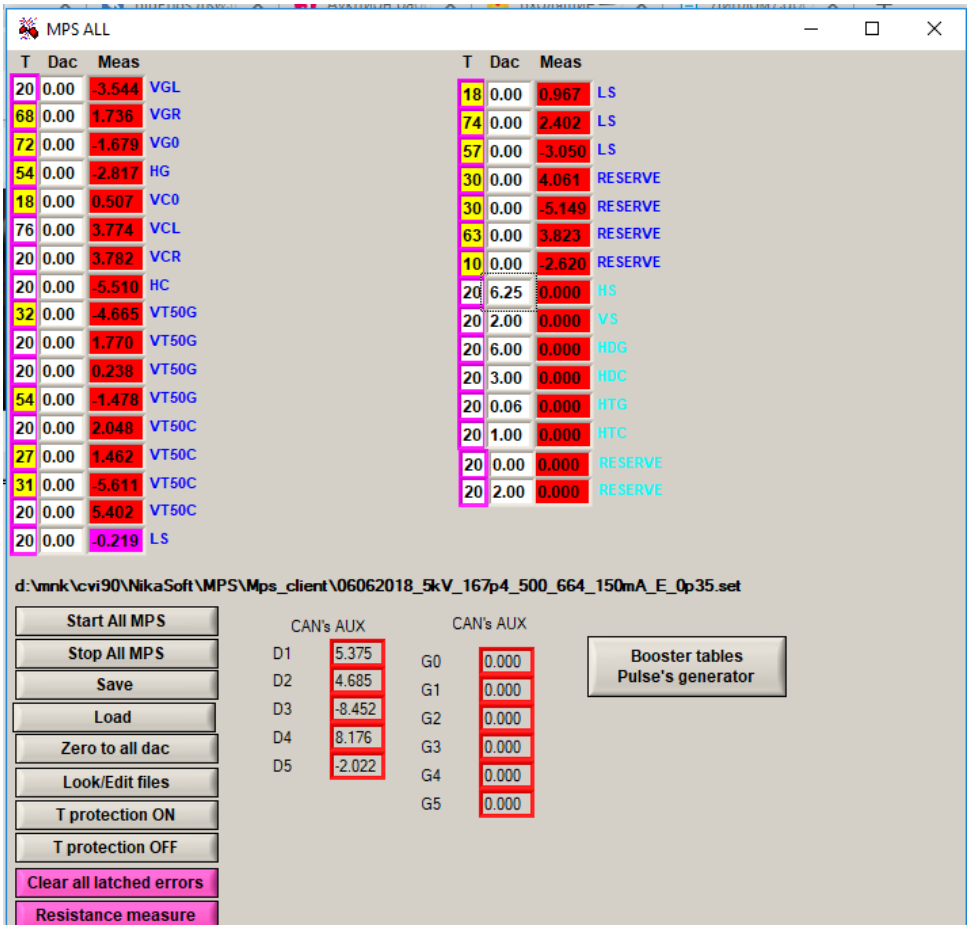

Рисунок 12. Взаимодействие с сервером и клиентом MPS

И затем была протестирована работа виртуальной системы с сервером IST. На рис. 13 представлено взаимодействие пяти виртуальных CEAC124 с 20 каналами АЦП, и 8 входными и выходными регистрами, который заменил CDAC20, в скрипте ist_script.py с клиентом и сервером IST, которые используются с действующей установкой электронного охлаждения ИЯФ СО РАН.

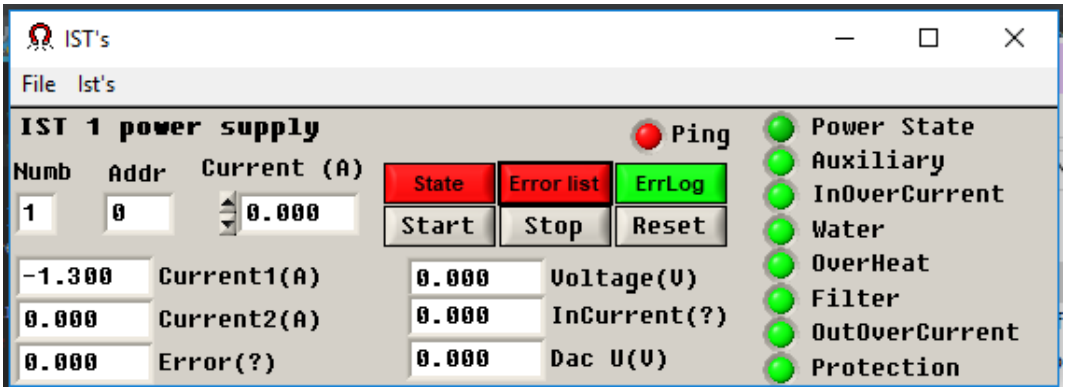

Рисунок 13. Взаимодействие с сервером и клиентом IST

Во время теста регистры устройств были выставлены в единицу, в результате клиент IST отобразил эти состояния регистров круглыми индикаторами справа, окрасив их в зеленый цвет.

5.3 Нагрузочное тестирование системы верификации

На один реальный порт CAN можно подключить до 64 устройств. В связи с этим, было проведено тестирование виртуальных портов с соответствующим количеством виртуальных устройств. Для этого подключалось по 64 виртуальных CEAC124 со случайно меняющимися значениями каналов АЦП на каждый из двух портов. Прежде всего был отправлен широковещательный запрос атрибутов устройств, а затем запуск измерения АЦП с запросом ответных посылок. Система показала свою работоспособность и корректность ответов от виртуальных устройств с помощью CanGwMonitor.

Затем было проведено тестирование скорости передачи в порт, к которому были подключены 64 виртуальных устройства CEDIO_A. Алгоритм отправки запросов представлен на рис. 14:

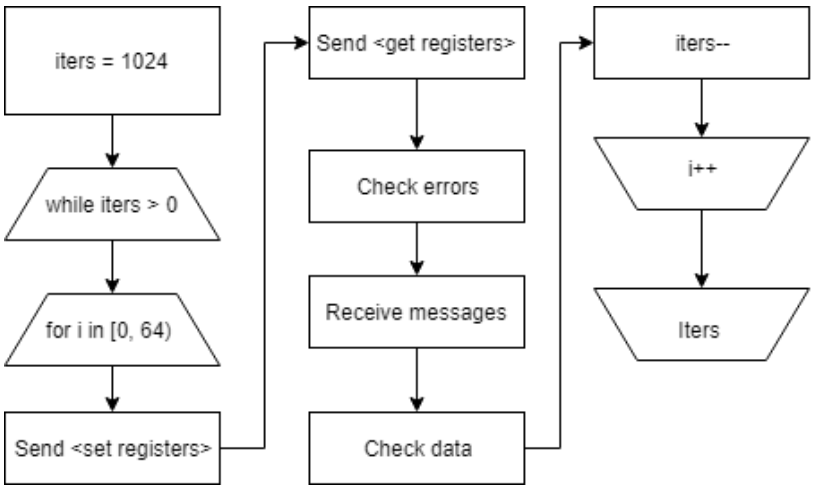

Рисунок 14. Алгоритм клиентской программы для тестирования

В результате теста было получено, что установленные клиентом значения регистров получены при запросе чтения регистров для каждого из 64 устройств и среднее время на две операции отправки запросов и одну операцию получения равно 9 мкс. TCP/IP клиент для данного теста был написан на языке C. Время замерялось с помощью функции clock().

6. Дальнейшее развитие системы

Основным компонентом данной системы верификации является TCP/IP ←→CAN шлюз. Очевидно, что этот шлюз используется не всегда и не на всех установках. Структура системы позволяет работать с виртуальными CAN устройствами и без этого шлюза за счет отдельного сервера, эмулирующего шину CAN. Чтобы взаимодействовать с виртуальными устройствами без шлюза, необходимо написать клиентское приложение, которое будет отправлять массив байт со структурой CAN пакета в виртуальную шину CAN. Так как структура пакета CAN известна, то единственный необходимый компонент такого клиентского приложения – это ZeroMQ request сокет. Через него будут отправляться запросы в виртуальную шину CAN в следующем формате: Первый байт – код запроса (0 – получить сообщения от устройств, 1 – отправить запрос устройствам). Следующие байты – запрос в формате CAN либо ничего, в случае запроса на получение пришедших сообщений. Хорошим решением было бы сделать класс-шаблон клиентского приложения для работы с устройствами без использования шлюза.

Использовать эту систему с другим протоколом взаимодействия (не CAN) невозможно ввиду следующих причин:

- Отличается формат пакетов – в данной системы виртуальная шина разбивает пришедшие от клиентов массивы байт (которые могут содержать несколько запросов) на отдельные запросы в соответствии со спецификацией протокола и отправляет их устройствам.
- Другой протокол может иметь не только широковещательный формат общения, но и адресный, соответственно виртуальная шина должна уметь отправлять сообщения конкретным устройствам.

Таким образом, разработанная система верификации не только позволяет работать с оборудованием, использующим интерфейс CANbus, с помощью TCP/IP ←→CAN шлюза, распространенного в ИЯФ СО РАН, но также позволяет писать собственные клиентские приложения для работы с виртуальными устройствами напрямую.

7. Заключение

В настоящее время промоделирован шлюз между CAN шиной и TCP/IP-клиентами, CAN шина, осуществляющая распределение CAN-сообщений, приходящих от клиентов и от виртуальных устройств, а также многопортовый регистр ввода/вывода CEDIO_A, устройства для управления и контроля напряжений источников питания в системах управления ускорительных комплексов, а также универсальные ЦАП/АЦП широкого применения CEAC208 и CEAC124, многопортовый регистр ввода/вывода с каналами ЦАП CEDIO_D и 20(40)-канальный прецизионный АЦП с входным/выходным регистрами CEAD20.

Для передачи сообщений между шлюзом, CAN шиной и виртуальными устройствами используется библиотека обмена сообщениями ZeroMQ, позволяющая организовать быстрый асинхронный обмен сообщениями с использованием на транспортном уровне таких протоколов как TCP, IPC, PGM и предоставляющая компромисс между сложностью реализации и высокой производительностью.

Работоспособность смоделированной системы протестирована с помощью такого программного обеспечения, использующегося в ИЯФ СО РАН и проекте NICA, как программы сервера и клиента системы защит (UBS), клиента и сервера источников питания корректирующих магнитов (MPS), клиента и сервера системы управления генератором тактирующих (SGF), клиента и сервера для создания продольного магнитного поля (IST) и специализированного анализатор трафика шины CAN(CanGwMonitor).

Во время тестов было выявлено, что на каждом из 2х портов может одновременно работать до 64 виртуальных устройств, так же, как на реальной CAN линии, и со среднем временем 9 мкс для двух операций отправки запросов и одной получения сообщений от виртуальных устройств.

8. Список литературы

[1] Егоров В. В. Методы верификации программного обеспечения / Н. И. Тамилова, А. Ж. Амиров, К. Н. Касылкасова // Молодой ученый – 2016, Вып. № 21(125) ч. 2 – С. 138-141.

[2] Кулямин В. В. Методы верификации программного обеспечения // Институт системного программирования РАН М.: Конкурс обзорно-аналитических статей по направлению «Информационно-телекоммуникационные системы», 2008. Режим доступа: http://www.ispras.ru/publishations/methods_of_software_verification.pdf (дата обращения 06.04.2019)

[3] Сравнение CAN и RS-485 / Под ред. Дэйтамикро, 2012. Режим доступа: http://www.datamicro.ru/download/CAN_vs_RS485[EA,%20rus].pdf (дата обращения 06.04.2019)

[4] Interface circuits for TIA/EIA-232-F / Texas Instruments Incorporated, 2002. http://www.ti.com/lit/an/slla037a/slla037a.pdf (дата обращения 06.04.2019)

[5] Эштенберг К. Сравнение CAN и Ethernet / Под ред. Дэйтамикро, 2006. Режим доступа: http://www.datamicro.ru/download/Comparison_CAN_and_Ethernet.pdf (дата обращения 06.04.2019)

[6] Козак В. Р. CEDIO_A, 2009. Режим доступа: http://www.inp.nsk.su/~kozak/designs/cedioa.pdf (дата обращения: 05.04.2019)

[7] Козак В. Р. CEAD20, 2009. Режим доступа: http://www.inp.nsk.su/~kozak/designs/cead20r.pdf (дата обращения: 05.04.2019)

[8] Козак В. Р. CEAC124, 2009. Режим доступа: http://www.inp.nsk.su/~kozak/designs/ceac124r.pdf (дата обращения 05.04.2019)

[9] Nitro2005 ZeroMQ: сокеты по-новому / Habr Запись от 5 ноября 2014. Режим доступа: https://habr.com/ru/post/242359/ (дата обращения 06.04.2019)

[10] What is Tango Controls? Режим доступа: http://www.tango-controls.org/what-tango-controls/ (дата обращения 05.04.2019)

[11] Vazquez-Otero Alejandro Tango vs EPICS technical comparison ELI Beamlines, 2014. Режим доступа: https://www.researchgate.net/publishation/299369316_Tango_vs_EPICS_technical_comparison_ELI_Beamlines (дата обращения 07.04.2019)

[12] CanGw User's guide, 2010. Режим доступа: http://www.inp.nsk.su/~mamkin/cangw_api/ (дата обращения 07.04.2019)

[13] Селиванов П. А. Разработка матобеспечения шлюза CANGW, применение в ИЯФ (система термоконтроля ЛСЭ): Квалификационная работа на соискание степени магистра. Новосибирский государственный университет, Новосибирск, 2005

[14] Козак В. Р. CAC208/CEAC208, 2010. Режим доступа: http://www.inp.nsk.su/~kozak/designs/cac208r.pdf (дата обращения 07.04.2019)

Приложение А. Примеры скриптов

Скрипт test_script.py

```
import CEDIO_A
from Executor import *
import zmq, threading

#create an event to work with threads in devices
event = threading.Event()

devices = [CEDIO_A.CEDIO_A(event=event, id=1,
                           random_regs_flag=True)]

#create poller to poll devices
poller = zmq.Poller()
my_executor = Executor(poller, devices, event)
my_executor.start()
```

Скрипт sgf_script.py

```
import CEDIO_D, CEAC124
from Executor import *
import zmq, threading

event = threading.Event()

ceac1e = CEAC124.CEAC124(event=event, id=0x1E)
ceac1f = CEAC124.CEAC124(event=event, id=0x1f)
ceac23 = CEAC124.CEAC124(event=event, id=0x23)
cedio24 = CEDIO_D.CEDIO_D(event=event, id=0x24)
cedio20 = CEDIO_D.CEDIO_D(event=event, id=0x20)

devices = [ceac1e, cedio24, cedio20, ceac1f, ceac23]

poller = zmq.Poller()
my_executor = Executor(poller, devices, event)
my_executor.start()
#set random values to ADC channels to test
for i in range(0,12):
    ceac1e.set_ADC_channel(i, 1.2)
    ceac1f.set_ADC_channel(i, 2.4)
    time.sleep(1)
    ceac23.set_ADC_channel(i, 0.5)
    time.sleep(1)
    cedio20.set_in_reg(1, 1)
```

Скрипт ubs_script.py

```
import CEDIO_A, CEAD20
from Executor import *
import zmq, threading

devices = []
event = threading.Event()

for i in range(17, 23):
    devices.append(CEDIO_A.CEDIO_A(event=event, id=i,
                                   random_regs_flag=True))

devices.append(CEAD20.CEAD20(event=event, id=23,
                random_ADC_flag=True))

poller = zmq.Poller()
my_executor = Executor(poller, devices, event)
my_executor.start()
```

Скрипт mps_script.py

```
import CEAC124, CEAC208
from Executor import *
import zmq, threading

devices = []
event = threading.Event()

for i in range(0x01, 0x09):
    devices.append(CEAC208.CEAC208(event=event, id=i,
                    random_ADC_flag=True))

for i in range(0x09, 0x11):
    devices.append(CEAC124.CEAC124(event=event, id=i,
                    random_ADC_flag=True))

poller = zmq.Poller()
my_executor = Executor(poller, devices, event)
my_executor.start()
```

Скрипт ist_script.py

```
import CEAC124
from Executor import *
import zmq, threading

def_send = "tcp://192.168.0.105:5556"
def_recv = "tcp://192.168.0.105:5557"

devices = []
event = threading.Event()

for i in range(0x19, 0x1D):
    devices.append(CEAC124.CEAC124(event=event, id=i,
send_addr=def_send, recv_addr=def_recv,
random_ADC_flag=True, dev_code=3, ADC_n=5, DAC_n=20,
in_reg=8, out_reg=8))

poller = zmq.Poller()
my_executor = Executor(poller, devices, event)
my_executor.start()

#change registers value for each working device
for i in range(0,8):
    for d in devices:
        d.set_in_reg(i,1)
```