НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

УТВЕРЖДАЮ

Зав. кафедрой Электроники

_____ Барбашов В.М.

«_____» _____ 2017 г.

ОТЧЁТ
О ДИПЛОМНОМ ПРОЕКТЕ

СИСТЕМА ЗАДАНИЯ ТЕМПЕРАТУРЫ ИС ПРИ ПРОВЕДЕНИИ РАДИАЦИОННЫХ ИСПЫТАНИЙ НА ВОЗДЕЙСТВИЕ ОЯЧ

(заключительный)

г. Москва, 2017

РЕФЕРАТ

Ключевые слова автоматизация, радиационные испытания, контроль температуры, микрокомпьютеры

Отчет содержит 111 стр. 16 рис. 13 таблиц.

Целью работы является разработка элементов автоматизированной системы удалённого контроля температуры СБИС при проведении радиационных испытаний.

В результате работы был разработан комплекс удалённого контроля температуры СБИС на основе ПИД-регулятора ТРИД РТП322 и микрокомпьютера Raspberry Pi 3.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

В настоящем дипломном проекте применяют следующие термины с соответствующими определениями:

Большое целое — целое число с неограниченной разрядностью.

Обратный прокси — тип прокси-сервера, который ретранслирует запросы клиентов из внешней сети на один или несколько серверов внутренней сети.

Одноплатный микрокомпьютер — маленький, относительно недорогой компьютер с микропроцессором в качестве CPU. «Одноплатный» означает, что микрокомпьютер реализован в виде одной печатной платы, без интегрированной периферии вроде клавиатуры либо экрана.

ТРИД — ПИД-регулятор температуры двухканальный ТРИД РТП322, разработанный ООО «Вектор-ПМ».

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ОЯЧ — отдельные ядерные частицы.

ИС — интегральная схема.

ЯП — язык программирования.

СБИС — сверхбольшая интегральная схема.

ПО — программное обеспечение.

ОС — операционная система.

API — application programming interface — внешний интерфейс взаимодействия с приложением.

PyPI — Python Package Index — официальный репозиторий с библиотеками Python.

ШИМ — широтно-импульсная модуляция.

ВВЕДЕНИЕ

Целью работы является разработка элементов автоматизированной системы удалённого контроля температуры СБИС при проведении радиационных испытаний. В настоящий момент контроль за температурой СБИС при проведении радиационных испытаний осуществляется посредством системы из находящихся в одном корпусе ПИД-регулятора ТРИД РТП322 и блока питания для нагревательных элементов, и находящихся вне корпуса нагревательных элементов совмещённых с измерительной термопарой. Настройка системы осуществляется вручную путём использования имеющейся у ТРИД лицевой панели, либо через подключённый по RS485 персональный компьютер с программой на ЯП LabVIEW.

Оба варианта управления не отличаются удобством: для использования лицевой панели необходимо находиться в месте проведения испытания, кроме того интерфейс из четырёх физических кнопок проигрывает по своим характеристикам (время задания параметра оператором, вероятность совершения ошибки оператором при задании параметра) возможному программному интерфейсу. Для использования персонального компьютера, подключённого по RS485 необходимо, чтобы указанный компьютер был размещён в месте проведения испытания и подключён к ТРИД. Указанный персональный компьютер может управляться удалённо, с использованием TeamViewer или аналогичного ПО, что приводит к использованию двух массивных блоков персональных компьютеров: одного в месте испытания и другого в операторской.

Таким образом, необходимо разработать систему контроля температуры, удовлетворяющую следующим требованиям:

а) Возможность доставки системы в/из места проведения испытания с минимальными усилиями.

б) Возможность быстрой установки в месте проведения испытания.

в) Возможность удалённого управления.

1 Аналитический раздел

Автоматизированная система управления ТРИД предназначена для испытаний микросхем с контролем температуры в установках (ускорительных комплексах) У-400 и У-400М лаборатории ядерных реакций имени Г.Н. Флерова. Данные установки представляют собой циклотроны, предназначенные для ускорения ионов с атомными массами в диапозоне $4 \div 209$. Тестирование происходит путём облучения микросхемы ионизирующим излучением (пучками □-частиц и тяжёлых ионов).

Тестируемая микросхема вместе с нагревательным элементом и датчиками температуры размещается в вакуумной камере (см. рис. 1.1), тогда как ТРИД с управляющей системой размещаются в отдельной стойке в специальном корпусе (см. рис. 1.2). В связи со стоимостью работы установки возможно одновременное тестирование нескольких микросхем, для чего используются разные каналы ПИД-регулятора: используемая модель ПИД-регулятора ТРИД РТП322 поддерживает до двух одновременно работающих каналов.

В месте размещения стойки с ТРИД существует возможность подключения к одноранговой локальной сети посредством либо WiFi (IEEE 802.11a/b/g), либо Ethernet (10/100/1000 Мбит/с, IEEE 80.23i/y/ab), другие способы подключения к локальной сети, а также связь с сетью Интернет не предусмотрены. Также непосредственный доступ оператора к стойке во время проведения испытаний не позволяется техникой безопасности в связи с наличием угрозы поражения радиацией. Поэтому комплекс, в котором проходят испытания, имеет два основных физических разделённых модуля:

а) место проведения испытаний, в котором находится камера для проведения испытаний и бóльшая часть оборудования

б) и операторская, в которой находятся компьютеры, предоставляющие доступ к оборудованию, размещённому в месте проведения испытаний.

К сожалению, все представленные на рынке модели ПИД-регуляторов оснащены либо только интерфейсом, предполагающим ручное управление, либо им и ещё одним из следующих интерфейсов: RS-485 и RS-232. Оба варианта требуют дополнительного оборудования для обеспечения удалённого управления, что и послужило основной причиной создания данного проекта.

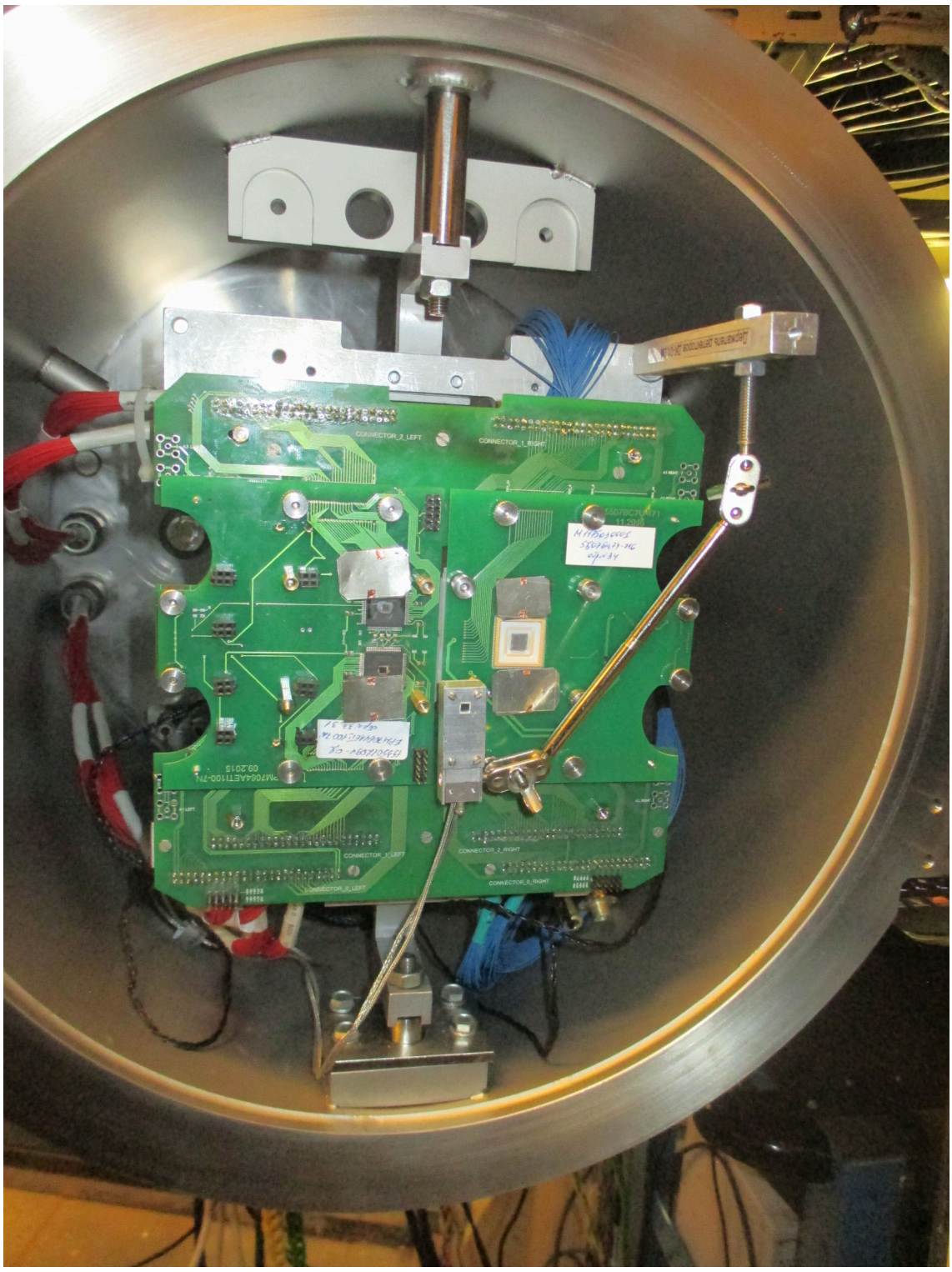


Рисунок 1.1 — Вакуумная камера с установленными микросхемами

Рисунок 1.2 — Корпус, в котором содержится ТРИД и управляющая система

2 Конструкторский раздел

2.1 Определение способа удалённого доступа и подбор типа управляющего модуля

Согласно разделу 1, единственным доступным способом удалённого управления является использование предоставляемой локальной сети, поэтому первым требованием к управляющему модулю является поддержка TCP/IP стека.

Выбор между подключением по WiFi и подключением по Ethernet определяется объёмом траффика между управляющим модулем и компьютером оператора, допустимой задержкой реакции системы на команды оператора и допустимой задержкой при передаче измеренных значений температуры. Рассмотрим этот вопрос подробнее:

а) Объём траффика — для контроля за ходом испытаний достаточно раз в секунду снимать показания температурного сенсора с ТРИД и иногда (примерно один раз в пять минут) присылать новые значения параметров. ТРИД поддерживает только 16-битовые целые, что даёт объём траффика от управляющего модуля не менее 32 бит в секунду. Установка параметров с определённым периодом не требуется, поэтому требования к сетевому подключению определяются не их объёмом, а только допустимой задержкой реакции на команды оператора.

б) Допустимая задержка реакции системы на команды оператора — одна секунда.

в) Допустимая задержка при передаче измеренных значений температуры составляет половину секунды. Данные значения записываются в журнал испытаний и необходимы для соотнесения колебаний температуры с отказами тестируемой микросхемы.

Как видно, при данных условиях возможно осуществление подключения как по WiFi сети, так и с использованием Ethernet. Однако подключение по WiFi сети облегчает процедуру сбора испытательного стенда, поэтому следующим требованием к управляющему модулю служит наличие возможности подключения по WiFi.

Также в связи с малыми объёмами траффика допустимо использование HTTP протокола и сериализация передаваемых данных не наиболее оптимальным, а наиболее удобным способом. Таким образом можно задействовать браузер для отображения интерфейса и взаимодействия с управляющим модулем, не тратя ресурсы на написание отдельного приложения.

ТРИД имеет единственный интерфейс подключения к управляющему модулю[1]: протокол modbus, работающий поверх UART через интерфейс RS485, что требует наличия RS485 у управляющего модуля. Также желательно наличие готовых библиотек для работы с modbus. В ходе исследования выяснилось, что ТРИД можно модифицировать для поддержки полнодуплексного UART по трём линиям: RX/TX/GND с высоким уровнем равным 5 В.

Таким образом, управляющий модуль должен соответствовать следующим требованиям:

а) Помещаться в одном корпусе с ТРИД (размеры корпуса: $205 \times 135 \times 435$ мм).

б) Поддерживать TCP/IP.

в) Поддерживать подключение по WiFi (802.11a/b/g) и Ethernet (802.3i/y/ab: 10/100/1000 Мбит/с по витой паре).

г) Поддерживать UART/RS485 либо по RX/TX/GND (5В).

д) Иметь достаточно ресурсов для работы Web-сервера.

е) Иметь возможность отладки при неисправности сетевого подключения.

Для выполнения данных требований были рассмотрены следующие варианты:

а) Создание проекта на основе готовой платы с микроконтроллером.

б) Создание проекта на основе собственной печатной платы с микроконтроллером.

в) Создание проекта на основе микрокомпьютера.

Данные варианты имеют следующие плюсы и минусы:

Вариант создания проекта на основе готовой платы с микроконтроллером

Плюсами данного варианта являются:

а) Наличие готовых библиотек для работы с периферией (к которой в данном варианте относятся как ТРИД, так и Ethernet и WiFi модули).

б) Более низкое энергопотребление по сравнению с микрокомпьютерами.

в) Остуствие необходимости разработки собственной печатной платы (по сравнению со вторым вариантом).

Минусы:

а) Невозможность удалённой отладки по сети в случае проблем с разрабатываемым программным комплексом, а не с библиотеками, обеспечивающими работу с сетью.

б) Необходимость разработки всего программного комплекса на языке низкого уровня.

в) Необходимость создания абстракции многозадачности для параллельной обработки сетевых запросов и опроса ТРИД собственными силами.

г) Отладка с помощью низкоуровневых интерфейсов (JTAG).

Для оценки дополнительной трудоёмкости данного варианта по сравнению с вариантом на основе микрокомпьютера были найдены готовые библиотеки и проекты, включение которых или аналогов в разрабатываемую прошивку необходимо. Найденный проекты представленные в табл. 2.1.

Таблица 2.1 — Метрики готовых библиотек для плат Arduino

Проект	Описание	SLOC[a]
avr-tasks[2]	Реализация многозадачности для Atmel AVR микроконтроллеров, используемых в платах Arduino[b].	1 087
ChatServer[3]	Пример реализации чата на основе сетевой библиотеки Arduino (Ethernet).	37
WiFiChatServer[3]	Пример реализации чата на основе сетевой библиотеки Arduino (WiFi).	53

[a] Число строк кода, за исключением комментариев и пустых строк.

[b] Согласно заявлению автора, может быть легко адаптирована для других микроконтроллеров.

Вариант создания проекта на основе собственной печатной платы с микроконтроллером

Плюсами данного варианта являются:

а) Наиболее низкое энергопотребление.

Минусы:

а) Невозможность удалённой отладки по сети в случае проблем с разрабатываемым программным комплексом, а не с библиотеками, обеспечивающими работу с сетью.

б) Необходимость разработки всего программного комплекса на языке низкого уровня.

в) Необходимость создания абстракции многозадачности для параллельной обработки сетевых запросов и опроса ТРИД собственными силами.

г) Необходимость создания либо адаптации библиотек для работы с периферией.

д) Отладка с помощью низкоуровневых интерфейсов (JTAG).

Для оценки дополнительной трудоёмкости данного варианта по сравнению с вариантом на основе микрокомпьютера были найдены готовые библиотеки и проекты, включение которых или аналогов в разрабатываемую прошивку необходимо. Найденный проекты представленные в табл. 2.2.

Вариант с созданием проекта на основе микрокомпьютера

Плюсы:

а) Возможность использования языков программирования высокого уровня.

б) Наличие высокоуровневых абстракций многозадачности.

в) Наличие высокоуровневых библиотек для доступа к периферии и сети.

Таблица 2.2 — Метрики готовых библиотек для плат Arduino

Проект	Описание	SLOC[a]
avr-tasks[2]	Реализация многозадачности для Atmel AVR микроконтроллеров, используемых в платах Arduino[b].	1 087
ESP8266AVRISP[4]	Библиотека поддержки WiFi модуля ESP8266 для Arduino.	647
avr-enc28j60[5]	Библиотека поддержки Ethernet контроллера ENC28J60.	696

[a] Число строк кода, за исключением комментариев и пустых строк.

[b] Согласно заявлению автора, может быть легко адаптирована для других микроконтроллеров.

г) Изоляция собственного программного обеспечения от ПО, обеспечивающего многозадачность и доступ к периферии и сети. Как следствие, возможность удалённой отладки при наличии проблем в собственном программном обеспечении.

д) Отладка с помощью периферии, используемой персональными компьютерами — клавиатуры и экрана. Также возможен прямой доступ к перезаписываемой памяти с «прошивкой» микрокомпьютера.

Минусы:

а) Высокое энергопотребление.

Таким образом, наиболее простым способом удовлетворить все требования является использование одноплатного микрокомпьютера с USB портами и/или поддержкой UART: ОС микрокомпьютера обеспечивает поддержку сетевых подключений и предоставляет простой доступ к UART, для использования UART/RS485 можно задействовать USB переходник, либо же модифицировать ТРИД.

2.2 Подбор управляющего микрокомпьютера

В соответствие с пунктом 2.1 управляющий микрокомпьютер должен

а) Помещаться в один корпус с ТРИД.
б) Блок питания для управляющего микрокомпьютера должен помещаться там же.
в) Иметь интерфейс USB (host), либо UART с RX/TX/GND и высоким уровнем 5 В.
г) Иметь интерфейсы WiFi (IEEE 802.11a/b/g) и Ethernet (IEEE 802.3i/y/ab).
д) Иметь достаточно ресурсов для работы Web-сервера.
е) Иметь возможность отладки при неисправности сетевого подключения.

Помимо этого желательно наличие хорошей официальной службы поддержки либо сообщества людей с опытом разработки устройств на основе выбранного микрокомпьютера; второе предпочтительнее т.к. доказывает жизнеспособность систем на основе данного микрокомпьютера.

В качестве кандидатов в управляющие микрокомпьютеры был рассмотрен ряд различных одноплатных микрокомпьютеров: см. таблицу 2.3. В рассмотрении учавствовали только микрокомпьютеры, соответствующие следующим параметрам:

а) стоимость пять тысяч рублей или ниже;

б) в наличии интерфейсы USB и UART для связи с ТРИД, последний должен быть дуплексным с логическими уровнями 5 В (единица) и 0 В (ноль);

в) в наличии интерфейсы WiFi и/или Ethernet. При наличии только одного предполагается возможность обеспечения подключения по другому интерфейсу с помощью устройств, подключающихся к USB;

г) количество памяти — 256 МиБ и больше;

д) имеется поддержка какого-либо дистрибутива на основе linux.

Среди рассмотренных микрокомпьютеров ODROID-C2, Intel Galileo и MB77.07 были отброшены из-за высокой стоимости при отсутствии встроенной поддержки WiFi. BeagleBone Black был отброшен из-за высокой стоимости при части характеристик ниже, чем у конкурентов и отсутствии либо встроенной поддержки WiFi, либо встроенной поддержки Ethernet. Pine A64+ был отброшен из-за наличия более дешёвой младшей модели Pine A64 с достаточными характеристиками для выполнения задачи.

Таблица 2.3 — Сравнительные характеристики микрокомпьютеров

Название микрокомпьютера	Цена, руб.	WiFi, 802.11*x*	Ethernet, *x*BASE-T	ОЗУ, МиБ	Долговременная память, макс.	Размеры В×Ш, мм
I	II	III	IV	V	VI	VII
ODROID-C2	4 500	через USB	10/100/1000	2 048	MicroSD/eMMC, 64 ГиБ	85×56
Pine A64+	1 700	b/g/n	10/100/1000	2 048	MicroSD, 256 ГиБ	133×80
Pine A64	1 000	b/g/n	10/100	512	MicroSD, 256 ГиБ	133×80
BeagleBone Black Rev C	3 400	b/g/n[1]	10/100	512	eMMC, 4 ГиБ	88×55
Banana Pi BPI-M1+	2 400	b/g/n	10/100/1000	1 024	MicroSD/SATA, 2 ТиБ	92×60
Intel Galileo Gen2	2 900	через USB	10/100	256	MicroSD, 32 ГиБ	124×72
Orange Pi Zero	1 000	b/g/n	10/100	256	TF/MMC, 64 ГиБ	48×46
Raspberry Pi 3	2 400	n	10/100	1 024	MicroSD, 64 ГиБ	85×56
MB77.07	4 800	через USB	10/100	256	встроенная	80×80

[1] Вместо Ethernet: микрокомпьютер может поставляться с Ethernet или WiFi, но не и с тем, и с другим.

Таким образом, по основным характеристикам наиболее подходящими являются Pine A64, Orange Pi Zero, Raspberry Pi 3 и Banana Pi BPI-M1+. Далее были проверены наличие и активность сообщества вокруг данных устройств. В ходе проверки были исследованы форумы, ссылки на которые имеются на официальных сайтах производителей данных микрокомпьютеров. Проверялось общее количество тем в подфорумах по выбранным моделям микрокомпьютеров. Также, в связи с тем, что форум Orange Pi Zero оказался неработоспособен, была добавлена метрика «Страниц в Google», получаемая путём поиска точного соответствия модели микрокомпьютера (т.е., к примеру, **"Orange Pi Zero"** для Orange Pi Zero). Полученные результаты указаны в табл. 2.4.

Таблица 2.4 — Результаты исследования сообществ микрокомпьютеров

Микрокомпьютер	Ссылка	Количество тем	Количество сообщений	Страниц в Google
Orange Pi Zero	[6]	N/A	N/A	235 000
Banana Pi BPI-M1+	[7]	207	476	6 490
Raspberry Pi 3[a]	[8]	55 904	317 128	564 000
Pine A64[b]	[9]	1 662	14 725	80 400

[a] Форум не разделён по моделям, данные даны по секции «Using the Raspberry Pi».

[b] Даны данные по подфорумам «General discussion on PINE A64(+)», «Linux», «Hardware, Accessories and POT» и «Projects, Ideas and Tutorials» секции «PINE A64(+)».

Исходя из этих данных было принято решение об исключении Banana Pi BPI-M1+ из рассмотрения вследствие недостаточно большой поддержки сообществом. Таким образом, было отобрано три наиболее подходящих микрокомпьютера — Orange Pi Zero, Pine A64 и Raspberry Pi 3, и принято решение, что ПО должно создаваться так, чтобы его можно было с минимальными усилиями адаптировать для всех трёх микрокомпьютеров. Для реализации проекта в настоящий момент был выбран Raspberry Pi 3 как имеющий наиболее активное сообщество.

2.3 Выбор ОС и ПО

В соответствие с пунктом 2.1 операционная система должна соответствовать следующими требованиями:

а) Работать на выбранном микрокомпьютере.
б) Потреблять минимальные ресурсы на собственные нужды.
в) Поддерживать контроллер UART и предоставлять к нему доступ.
г) Поддерживать TCP/IP, иметь возможность написания собственного сервера.
д) Поддерживать удалённый доступ для отладки сервера и загрузки ПО.

е) Поддерживать все три выбранных микрокомпьютера.

Согласно официальному сайту Raspberry Pi[10] единственной официально поддерживаемой ОС является Raspbian, однако существует возможность установки альтернативных ОС. Raspbian является дистрибутивом GNU/Linux, основанном на Debian, что позволяет предположить, что ПО под этот дистрибутив будет с минимальными модификациями работать в других дистрибутивах, основанных на Debian. Так как все три выбранных микрокомпьютера поддерживают либо Debian (Pine A64, Orange Pi Zero), либо Raspbian (Raspberry Pi 3, Orange Pi Zero), то Raspbian удовлетворяет всем требованиям и необходимость лишаться официальной поддержки отсутствует.

Raspbian поставляется в двух вариантах: большой образ с PIXEL и минимальный образ без графического интерфейса[11]. В связи с тем, что для отладки возможно использование удалённого доступа по ssh, в минимальном образе также доступен framebuffer, служащий заменой графическому интерфейсу для отладки системы при физическом доступе, а основным способом доступа к управляемой системе был выбран Web-интерфейс, то необходимость использования большого образа отсутствует.

Далее, к управляемой системе предполагается доступ одного оператора с одного компьютера в единицу времени. Требуется также параллельное автоматизированное снятие показаний температуры из программы на языке LabView на том же компьютере. Вследствие этого была выбрана следующая архитектура: непосредственно с локальной сетью связан nginx, предоставляющий статические файлы для работы web интерфейса, а также служащий в качестве обратного прокси для специальной программы, предоставляющей доступ к ТРИД (см. рисунок 2.1). «Специальная программа» представляет собой Web-сервер с API, основанным на JSON и может быть использована из LabView без взаимодействия с Web-интерфейсом.

В указанной конфигурации JSON был выбран из-за широкой поддержки в различных языках программирования, в том числе в браузере. Nginx представляет собой широко используемый и лёгкий в настройке Web-сервер[12]. Обновление данных на странице, представляющей собой набор статических файлов возложено на JavaScript для снижения требований к Web-серверу.

Для написания API backend был выбран язык Python: в число библиотек под этот язык входят библиотеки для работы с последовательным интерфейсом, библиотеки для создания Web сервисов, а также библиотеки для работы с протоколом modbus. Помимо этого, язык имеет безопасную модель памяти и обеспечивает быструю разработку за счёт быстродействия конечной программы. Для случая, если быстродействие недостаточно для нормальной работы, Python позволяет переписывать часть программы на других языках, поддерживающих те же соглашения о вызове функций, что и ЯП C.

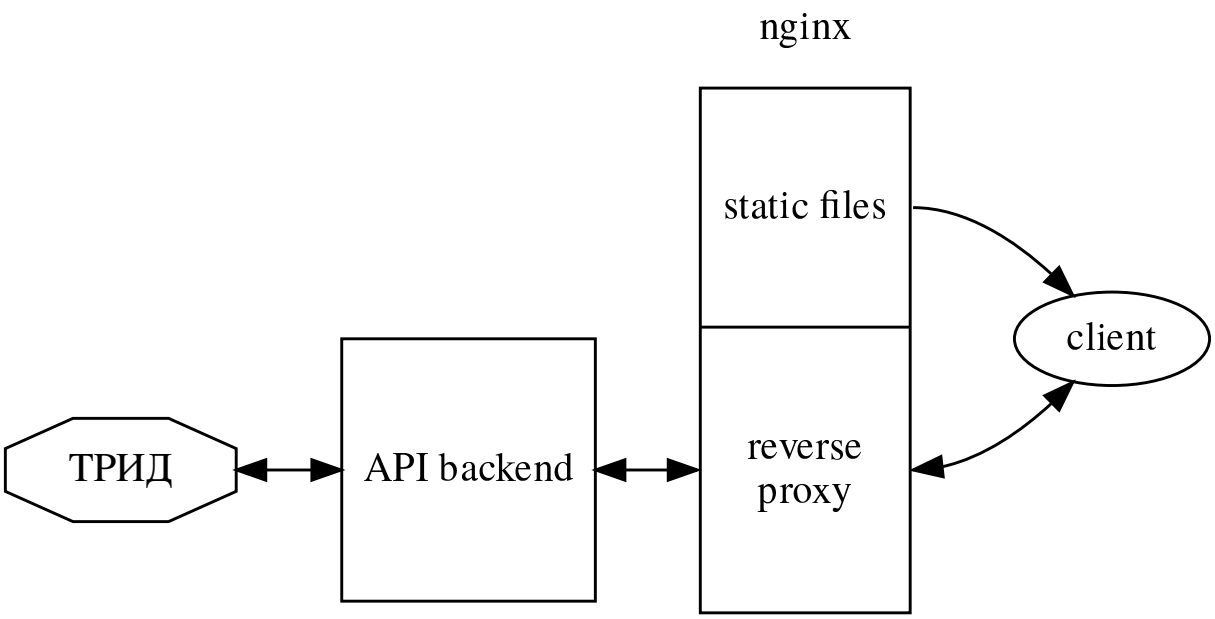


Рисунок 2.1 — Архитектура Web-сервиса

2.4 Выбор библиотек Python

Из предыдущего раздела видно, что для работы API backend требуются следующие библиотеки:

а) Библиотека для работы с последовательным портом ввода-вывода.
б) Библиотека для работы с протоколом modbus (поверх последовательного порта).
в) Библиотека для сбора отладочной информации (создания журнала).
г) Библиотека для создания Web сервиса.

В случае с библиотекой для работы с последовательным портом ввода-вывода есть фактически единственная альтернатива — pyserial[13], других развивающихся проектов данной тематики найти не удалось.

Протокол modbus поддерживается следующими библиотеками: MinimalModbus[14], pylibmodbus[15], modbus_tk[16], pymodbus[17], uModbus[18]. Из них pylibmodbus была отброшена из-за требования наличия дополнительной библиотеки на C, длительного отсутствия обновлений и отсутствия документации. Modbus_tk также отличалась отсутствием документации, а официальный пакет pymodbus в PyPI не поддерживается, хотя сама библиотека развивается[19]. Среди оставшихся MinimalModbus и uModbus первая была выбрана за более удобный интерфейс.

В качестве библиотеки для сбора отладочной информации оказалось возможным использовать часть стандартной библиотеки Python, предназначеную для этой цели. На nginx была дополнительно возложена обязанность предоставления создаваемых данной библиотекой журналов.

Среди библиотек, подходящих для создания Web интерфейса обнаружилось наибольшее разнообразие. Библиотека circuits[20] была выбрана из-за того, что для её использования требуется написать минимальное количество кода, а также отсутствие требований к файловой структуре проекта.

2.5 Выбор библиотек для Web-интерфейса

В соответствии с пунктом 2.3 Web-интерфейс преставляет из себя набор статических файлов, динамических изменяемые данные получаются путём использования API из JavaScript сценариев с последующим изменением страницы. Для предоставления доступа к большинству возможностей ТРИД таким образом достаточно относительно простого JavaScript кода: для работы Web-интерфейса необходимо

а) Отправлять API запросы на получение данных и получать ответы на них.
б) Изменять отображаемую страницу в соответствии с полученными ответами.
в) Получать от пользователя параметры работы ТРИД.
г) Отправлять полученные параметры в виде API запросов.
д) Отображать график изменения температуры во времени.

Для первой и четвёртой задач браузер предоставляет встроенное API **XMLHttpRequest**. Для второй задачи браузер предоставляет т.н. DOM API, позволяющее не просто вставить полученные значения в предопределённые места, но и полностью изменить внешний вид страницы при необходимости. Для третьей задачи существуют различные события и, опять же, DOM API. Но вот удобного способа рисования графиков, который был бы встроен в браузер нет. Таким образом, необходимо найти библиотеку для отображения графиков на JavaScript, удовлетворяющую следующим требованиям:

а) Позволяет отображать графики временных рядов.
б) Позволяет обновлять графики в реальном времени.
в) Бесплатна для коммерческого использования.
г) Отображает сетку и позволяет добавлять свои линии.
д) Активно разрабатывается.
е) Имеет подробную документацию.

Поиск показал, что JavaScript библиотек, удовлетворяющих первым двум требованиям довольно много. Выбор был остановлен на первой библиотеке из найденных, которая удовлетворяла всем указанным требованиям: Dygraph.js[21].

3 Технологический раздел

3.1 Архитектура приложения

Как было показано в пункте 2.3, приложение разбито на два основных модуля: nginx и API backend. В данном разделе будет рассмотрена только архитектура API backend, настройка nginx рассматривается отдельно в пункте 3.2 а архитектура этой части тривиальна и фактически отображена на рисунке 2.1.

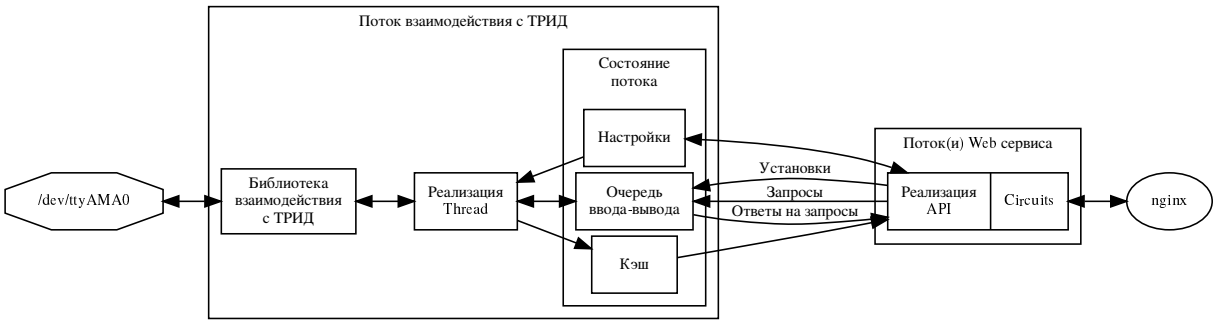


Рисунок 3.1 — Архитектура API backend

Архитектура API backend изображена на рис. 3.1. Всё приложение было разбито на три основных модуля: выделенная библиотека взаимодействия с ТРИД (используется внутри потока ввода-вывода), реализация потока ввода-вывода на основе классов из стандартной библиотеки Python — модуля **threading** и собственно реализации API на основе классов из библиотеки **circuits**.

Такая архитектура была создана из-за наличия только одной дуплексной линии связи с ТРИД: необходимо обеспечить строгую последовательность отправки запросов к ТРИД и получения ответов на случай прихода нового запроса от nginx во время обработки старого. Данную проблему можно решить, как минимум, на трёх уровнях: запретить nginx создавать более одного параллельного соединения к Web серверу, не использовать потоки при обработке запросов или работать с ТРИД исключительно через отдельный поток и очередь и/или блокировки. Третий вариант был выбран, из-за того, что существуют запросы, одновременная обработка которых безопасна. В основном это относится к запросам, получающим значения из кэша или настроек. Помимо этого поток взаимодействия с ТРИД требуется, чтобы постоянно опрашивать ТРИД, узнавая показания температурного датчика; без отдельного потока эту задачу пришлось бы возложить на Web-интерфейс, что привело бы к его замедлению.

3.2 Настройка nginx и circuits

В соответствие с рис. 2.1 nginx должен выполнять две функции: предоставлять доступ к статическим файлам (в них определяется весь Web-интерфейс) и предоставлять

доступ к API backend. Для этого достаточно небольшой конфигурации, показанной в листинге 1.

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;

    index index.html;

    server_name _;

    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ =404;
    }

    location /api {
        proxy_pass http://127.0.0.1:20000;
    }
}
```

Листинг 1 — Настройка Nginx

Данный файл настроек помещается в каталог **/etc/nginx/sites-enabled** под именем **default**.

В нём nginx настроен слушать на порту 80 (IPv4 и IPv6), брать статические файлы из /var/www/html и перенаправлять запросы с путём /api (к примеру, **http://mypihost/api/trid/monitor/get**) серверу (API backend) на порту 20 000. Для этого приложение Circuits должно выглядеть соответственно листингу 2.

```
from circuits.web import Controller, Server

class DummyAPI(Controller):
    '''Entry point for dummy API

    Handles requests to /api/dummy.

    :param dict common_state: Some common state.'''
    channel = '/api/dummy'

    def __init__(self, common_state, *args, **kwargs):
        super(DummyAPI, self).__init__(*args, **kwargs)
        self.__common_state = common_state
        common_state['dummy_counter'] = 0

    def increment(self):
        '''/api/dummy/increment request handler'''
        common_state['dummy_counter'] += 1
        return str(common_state['dummy_counter'])

class API(Controller):
    '''API entry point, serves nginx /api requests'''
    channel = '/api'
    '''Specify that this works with /api.'''

    def __init__(self, *args, **kwargs):
        super(API, self).__init__(*args, **kwargs)
        common_state = {}
        self += DummyAPI(common_state)


if __name__ == '__main__':
    application = Server(('127.0.0.1', 20000)) + API()
    aplication.run()
```

Листинг 2 — Настройка Circuits

Здесь создаётся простое приложение — Web-сервер, слушающий на порту 20 000 и предоставляющий всего одну возможность — узнавать, сколько раз был вызван API метод **/api/dummy/increment**. Порт, на котором сервер будет слушать сервер определяется в аргументах **Server**, атрибут **channel** служит для определения префикса метода(ов), которые будет обслуживать класс. Таким образом, **/api/dummy/increment** будет обслуживаться классом с атрибутом **channel**, равным **/api/dummy** и имеющим метод **increment**,

который и будет вызван при запросе **/api/dummy/increment**. **application.run()** в самом конце файла запускает бесконечный цикл, ждущий и обрабатывающий запросы.

3.3 Настройка автоматического запуска nginx и API backend

Одной из особенностей разрабатываемой системы являются частые перезагрузки, поэтому весьма желательно запускать nginx и API backend автоматически при старте системы. Для выполнения этой задачи можно использовать возможности дистрибутиве Raspbian, а именно систему инициализации systemd. Настройки, описывающие как система инициализации должна запускать nginx уже присутствуют в системе в качестве части пакета nginx, нужно только включить автозапуск: **sudo systemctl enable nginx**. Однако systemd не обладает информацией о способе запуска созданного API backend. Для исправления данной проблемы необходимо поместить в **/etc/systemd/system** файл **trid_site.service** с содержанием, указанным в листинге 3 и включить автозапуск: **sudo systemctl enable trid_site**.

```
[Unit]
Description=TRID API service
Wants=nginx.service
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/python /var/www/trid_site/main.py
PIDFile=/var/run/trid.pid
Restart=always
WorkingDirectory=/var/www

[Install]
WantedBy=multi-user.target
```

Листинг 3 — Настройка systemd

Здесь указывается, что сервис (API backend) требует для работы nginx (**Wants=nginx.service**), а для работы сервиса нужно просто запустить **/usr/bin/python /var/www/trid_site/main.py** в каталоге **/var/www**. Образец того, как должен выглядеть **/var/www/trid_site/main.py** есть в листинге 2.

3.4 Создание библиотеки взаимодействия с ТРИД

Библиотека взаимодействия с ТРИД абстрагирует низкоуровневый доступ к ТРИД, скрывая детали реализации протокола modbus, используемые адреса регистров и разбор ответа от ТРИД. Библиотека также позволяет проверить корректность введённых пользователем данных. Однако соответствие значениям из табл. В.4 не проверяется: за попыткой ввода значений, выходящих за данные пределы, не следует отказ ТРИД, поэтому эти значения сохранены только в Web-интерфейсе.

Библиотека предоставляет один класс для доступа к ТРИД (названный **TRID**) и набор констант для проверки пользовательского ввода (константы, начинающиеся с **SUPPORTED_**), а также набор констант со значениями по-умолчанию (константы, начинающиеся с **DEFAULT_**). Библиотека зависит только от стандартной библиотеки Python, pyserial[13] и MinimalModbus[14]. Сама библиотека называется **trid**.

Полное определение API библиотеки есть в приложении А.

3.4.1 Настройка последовательного порта

Библиотека MinimalModbus предполагает использование pyserial для доступа к порту, а именно являющегося частью библиотеки класса **serial.Serial**. Однако API MinimalModbus предполагает, что при инициализации класса **minimalmodbus.Instrument** будет передан единственный аргумент **serial.Serial** — **port**, то есть путь к последовательному порту (к примеру, **/dev/ttyAMA0**). Дополнительные настройки (скорость, чётность, размер слова, …) необходимо устанавливать, используя атрибут **Instrument.serial**.

Помимо этого, ТРИД предоставляет ограниченное количество комбинаций чётности, числа конечных бит и размера одного слова, из-за чего эти три настройки были объединены в один класс **trid.PortSettings**, а возможные комбинации сохранены в **trid.SUPPORTED_PORT_SETTINGS**. Таким образом, код настройки соединения с ТРИД, был прописан в методе инициализации класса и представлен в листинге 4.

```
class TRID(object):
    '''PID regulator interface

    May be used like this: ``with TRID('/dev/ttyAMA0') as t: ...``, this
    will automatically close /dev/ttyAMA0 at __exit__. It may also be
    closed with :py:meth:`TRID.close`.

    :param str tty: Terminal to work with.
    :param str mode: Mode: 'asc' or 'rtu'.
    :param int slaveaddress:
        PID regulator address. May be from 0x01 to 0xFF.
    :param int baudrate: PID regulator baud rate. Must be one of the values
        in SUPPORTED_BAUDRATES.
    :param PortSettings port_settings: PID regulator port settings, see
        :py:class:`PortSettings`.
    :param float timeout: Maximum time to wait when reading, in seconds.'''
    def __init__(self, port, mode=DEFAULT_MODE,
        slaveaddress=DEFAULT_SLAVE_ADDRESS, baudrate=DEFAULT_BAUD_RATE,
        port_settings=DEFAULT_PORT_SETTINGS, timeout=DEFAULT_TIMEOUT):

        assert(mode in MODE_TRANSLATIONS)
        assert(0x01 <= slaveaddress <= 0xFF)
        assert(baudrate in SUPPORTED_BAUDRATES)
        assert(port_settings in SUPPORTED_PORT_SETTINGS)
        self.mode = mode
        '''Currently effective mode'''
        self.slaveaddress = slaveaddress
        '''Currently used slave address'''
        self._instrument = Instrument(port, slaveaddress,
                                      MODE_TRANSLATIONS[mode])
        '''MinimalModbus interface.'''
        instr = self._instrument
        instr.serial.baudrate = baudrate
        instr.serial.parity = port_settings.parity
        instr.serial.bytesize = port_settings.bytesize
        instr.serial.stopbits = port_settings.stopbits
        instr.serial.timeout = timeout
```

Листинг 4 — Код инициализации класс TRID

3.4.2 Протокол взаимодействия с ТРИД

Согласно инструкции, ТРИД использует протокол modbus, однако некоторые детали не описаны в инструкции[1]. Управление ТРИД осуществляется путём установки

16-битных регистров, перечисленных на стр. 22. Значения в данных регистрах предоставляют из себя 16-битные знаковые целые, представленные в виде дополненного кода. Некоторые из этих регистров являются целыми числами, полученными из рациональных путём умножения на $0{,}1$, в таблице данный факт отражено в столбце «единицы измерения» (к примеру, $0{,}1$ °C). При запросе регистров допустимо запрашивать два регистра одновременно, если их адрес отличается не более, чем на единицу. Таким свойством обладают только регистры одинакового назначения, но для разных каналов: к примеру, «измеренное значение, канал 1» по адресу 0x0000 и «измеренное значение, канал 2» по адресу 0x0001.

При использовании MinimalModbus для реализации данного протокола достаточно четырёх вспомогательных функций: двух для конвертации 16-битных целых в/из двоичного кода, показанных в листинге 5 и двух методов класса.

```
def from_twos_complement(u16):
    '''Convert 16-bit unsigned integer to 16-bit signed integer'''
    mask = 2 ** (16 - 1)
    return -(u16 & mask) + (u16 & ~mask)

def to_twos_complement(i16):
    '''Convert 16-bit signed integer to 16-bit unsigned integer'''
    if i16 >= 0:
        return i16
    else:
        mask = (2 ** 16) - 1
        return (-1 - i16) ^ mask
```

Листинг 5 — Функции конвертации знаковых 16-битных целых в/из 16-битных беззнаковых целых чисел

Функции конвертации 16-битных чисел предполагают, что MinimalModbus воспринимает значения как 16-битные беззнаковые целые, сохранённые в стандартном типе **int**, тогда как входные значения представляют из себя 16-битные знаковые целые, также сохранённые в этом типе, однако не предполагая определённого формата их представления. При этом стандартный тип **int** представляет из себя большое целое число, поддерживающий битовые операции.

```
class TRID(object):
    # …
    def _get_ch_regs(self, address, channel, decimals=1):
        assert(channel in {None, 1, 2})
        self._instrument.serial.flushInput()
        if channel is None:
            ch1, ch2 = self._instrument.read_registers(address, 2)
            return (from_twos_complement(ch1) / (10 ** decimals),
                    from_twos_complement(ch2) / (10 ** decimals))
        else:
            address = address + channel - 1
            return from_twos_complement(self._instrument.read_register(
                address + channel - 1)) / (10 ** decimals)

    def _set_ch_regs(self, values, address, channel, decimals=1):
        assert(channel in {None, 1, 2})
        if channel is None:
            v1, v2 = (to_twos_complement(int(v * (10 ** decimals)))
                      for v in values)
            v = [v1, v2]
            return self._instrument.write_registers(address, v)
        else:
            address = address + channel - 1
            return self._instrument.write_register(
                address, to_twos_complement(values * (10 ** decimals)))
```

Листинг 6 — Реализация протокола взаимодействия с ТРИД на основе MinimalModbus

В листинге 6 показаны приватные методы класса **TRID**, используемые для реализации публичных методов. Вследствие их использования все публичные методы класса **TRID** состоят из единственного вызова одного из приватных методов с правильным адресом регистра, значением decimals (определяющим, сколько раз 16-битное целое нужно разделить на 10 для получения правильного значения: в большинстве случаев один раз, в остальных — 0) и предоставленными пользователем остальными аргументами.

3.5 Поток взаимодействия с ТРИД

Поток взаимодействия с ТРИД отвечает за синхронную обработку запросов к ТРИД, а именно, запросов на получение текущих значений настроек и их установку, а также за периодический опрос ТРИД для получения показаний температурных датчиков. Поток разделён на три основные части: библиотеку взаимодействия с ТРИД, описан-

ную в пункте 3.4, класс состояния потока, и собственно сама реализация потока на основе класса **threading.Thread** из стандартной библиотеки Python.

3.5.1 Класс состояния потока

Класс состояния потока содержит

а) очередь заданий (**Queue.Queue** из стандартной библиотеки), в которую поступают все запросы на взаимодействие с ТРИД, за исключением периодического опроса ТРИД и запросов на установку целевой температуры,

б) экземпляр класса **trid.TRID**, являющийся интерфейсом библиотеки взаимодействия,

в) информацию о целевой температуре, включая состояния «нагрев включён/отключён» раздельно для обоих каналов,

г) настройки опроса: интервал опроса и максимальное количество сохраняемых значений,

д) собственно сохранённые показания температурных датчиков,

е) блокировки:

1) совместная/исключительная блокировка для предотвращения получения несогласованных данных при запросе показаний датчиков (реализация блокировки взята из [22]),

2) исключительная блокировка (**threading.Lock** из стандартной библиотеки), предотвращающая переподключение к ТРИД во время его опроса,

3) исключительная блокировка (**threading.Lock**) для предотвращения получения несогласованных установок (настроек опроса и целевой температуры);

ж) событие (**threading.Event** из стандартной библиотеки) остановки, позволяющее мягко завершить поток,

з) вспомогательные методы для подключения к ТРИД, сохранения настроек, которые также сохраняются в долговременую память и установки/получения целевой температуры.

Основные задачи класса — хранение состояния потока, через которое также обеспечивается взаимодействие с реализацией API на основе circuits. Методы класса не абстрагируют использование блокировок, взаимодействие с очередью и событием или получение данных.

Метод инициализации класса приведён в листинге 7, вспомогательные функции — в листинге 8.

```
import threading
import Queue
class PIDControllerThreadState(object):
    def __init__(self, port_name='pi-uart'):
        self.shutdown_event = threading.Event()
        self.job_queue = Queue.Queue()
        self.data_lock = RWLock()
        '''RWLock used for exported data'''
        self.data = PIDControllerData()
        '''Data read/written to by the thread'''

        self.trid = None
        self.trid_port = port_name
        self.trid_lock = threading.Lock()

        self.interval = 1.0
        self.data_points = 100
        self.target_enabled = (None, None)
        self.target_temp = (None, None)
        '''Last set target temperature'''
        self.did_set_target_temp = False
        '''True if setting target temperature is not needed'''
        self.need_save_settings = True
        '''True if saving settings is needed'''
        self.state_lock = threading.Lock()

        self.trid_connect(port_name)
        self.save_settings()
```

Листинг 7 — Метод инициализации состояния потока

```
DISABLED_TEMP = -200
'''Temperature set when disabling heating.'''

class PIDControllerThreadState(object):
    # …
    def trid_connect(self, port_name, **kwargs):
        '''(Re)connect to PID regulator

        Keyword arguments are passed directly to :py:class:`trid.TRID`.

        :param str port_name: Port name, key in PORTS global.'''
        target_temp = (None, None)
        try:
```

```
            with open(SAVED_SETTINGS_FILE, 'r') as fp:
                settings = json.load(fp)
        except IOError:
            pass
        else:
            with self.state_lock:
                target_temp = tuple(settings['target_temp'])
                self.interval = settings['interval']
                self.data_points = settings['data_points']
        with self.trid_lock:
            if self.trid is not None:
                self.trid.close()
            self.trid = trid.TRID(PORTS[port_name], **kwargs)
        with self.state_lock:
            self.set_target_temp(target_temp, (False, False))
        self.save_settings()

    def save_settings(self):
        '''Save settings: target temperature, interval, update time

        Saves to SAVED_SETTINGS_FILE.'''
        if not self.need_save_settings:
            return
        with self.state_lock:
            settings = {
                'target_temp': self.target_temp,
                'interval': self.interval,
                'data_points': self.data_points,
            }
            self.need_save_settings = False
        try:
            with open(SAVED_SETTINGS_FILE, 'w') as fp:
                json.dump(settings, fp)
        except Exception:
            logger.exception('Failed to save settings')
            self.need_save_settings = True

    def set_target_temp(self, target_temp, target_enabled):
        '''Set target temperature and record it in the file

        Must be called under :py:attr:`state_lock`.

        File to record in: SAVED_SETTINGS_FILE.'''
        target_temp = (
          first_non_none(target_temp[0],self.target_temp[0],DISABLED_TEMP),
```

```
          first_non_none(target_temp[1],self.target_temp[1],DISABLED_TEMP),
        )
        target_enabled = (
          first_non_none(target_enabled[0], self.target_enabled[0], False),
          first_non_none(target_enabled[1], self.target_enabled[1], False),
        )
        self.target_enabled = target_enabled
        self.target_temp = target_temp
        self.need_save_settings = True
        self.did_set_target_temp = False

    def get_target_temp(self):
        '''Get effective target temp (what will be transferred to TRID)

        Must be called under :py:attr:`state_lock`.'''
        target_temp = self.target_temp
        enabled = self.target_enabled
        return (
            (target_temp[0] if enabled[0] else DISABLED_TEMP),
            (target_temp[1] if enabled[1] else DISABLED_TEMP),
        )
```

Листинг 8 — Вспомогательные функции класса состояния потока

3.5.2 Реализация потока взаимодействия с ТРИД

Основной частью реализации потока взаимодействия с ТРИД является потомок класса **threading.Thread** из стандартной библиотеки Python. Именно в этой части реализован периодический опрос и синхронная обработка заданий из очереди.

Поток взаимодействия с ТРИД действует следующим образом:

а) Получает настройки от класса состояния (блок-схема на рис. Б.1).

б) Если в соответствии с настройками требуется установка целевой температуры ТРИД, устанавливает целевую температуру (блок-схема на рис. Б.2).

в) Получает и сохраняет показания температурных датчиков (блок-схема на рис. Б.3).

г) Обрабатывает задачи, полученные от оператора (блок-схема на рис. Б.5).

Задача от оператора обрабатывается, если она либо уже есть в очереди, либо если она пришла за время, не превышающее интервал между опросами ТРИД с учётом того, что предыдущие шаги уже заняли некоторое количество времени.

При наличии задач от оператора, но отсутствии времени (к примеру, если интервал между опросами установлен в ноль), обрабатывается ровно одна задача от оператора на одно считывание показаний температурных датчиков.

Подробная блок-схема потока приведена в приложении Б, код потока есть в листинге 37.

3.6 Создание API

В соответствие с табл. А.3, по UART доступны следующие возможности ТРИД: получение/установка ширины гистерезиса, получение/установка значений аварийной температуры (3 канала), установка коэффициентов для ПИД-регулирования (K_p, K_i, K_d), получение/установка целевой температуры, получение показаний температурных датчиков. В дополнении к этому API поддерживает получение показаний температуры за данный период (все сохранённые показания, либо показания температуры, снятые после указанного времени), сброс сохранённых показаний, установку параметров подключения к ТРИД, установку максимального числа хранимых показаний температуры и частоты опроса ТРИД.

Для облегчения работы с ТРИД API получения/установки целевой температуры поддерживает состояния «нагрев отключён» (ТРИД в этом случае используется только для снятия показаний) и «нагрев включён». Данные состояния реализуются через указание минимальной (-200 □) температуры в качестве целевой, таким образом не предполагая использования ТРИД с подключением к охлаждающему устройству помимо нагревающего. В табл. В.1 перечислены все API методы.

Также API backend сохраняет информацию о параметрах подключения, параметрах снятия показаний (интервал, сохранённое количество) и последней использованной целевой температуре в файловой системе управляющего микрокомпьютера. Независимо от сохранённой целевой температуры при запуске сервиса включается режим «нагрев отключён».

В ходе проверки разработанного приложения было выяснено, что использование каналов аварийной сигнализации, за исключением канала А, ведёт к отказу ТРИД: в зависимости от способа использования ТРИД либо немедленно перезагружается, либо пишет в канал связи данные, не соответствующие протоколу modbus. Поэтому API сервиса не поддерживает установку каналов аварийной сигнализации, отличных от А.

3.6.1 Описание API на основе классов circuits

Как несложно увидеть, значительная часть методов из таблицы В.1 представляет из себя просто пару из получения метода и его установки. Все данные методы реализуются на основе общего предка **TRIDGetSetAPI**, показанного в листинге 9.

```
import circuits.web.exceptions as cwe
from circuits.web import Controller

class TRIDGetSetAPI(Controller):
    '''/api/*/{get,set} API entry point

    :param PIDControllerThreadState thread_state:
        PID controller thread state.'''
    def __init__(self, thread_state, *args, **kwargs):
        super(TRIDGetSetAPI, self).__init__(*args, **kwargs)
        self.__thread_state = thread_state
    def get(self, channel=None):
        if channel not in {None, '1', '2'}:
            raise cwe.BadRequest(json.dumps(
                {'error': 'Unexpected channel'}))
        return request(self.__thread_state, 'get_' + self._trid_suffix,
                       channel, **self._trid_kwargs)
    def POST(self, value1=None, value2=None):
        if value1 is None and value2 is None:
            raise cwe.BadRequest(json.dumps({
                'error': 'Must supply at least one value'}))
        try:
            if value1 is not None: value1 = float(value1)
            if value2 is not None: value2 = float(value2)
        except ValueError:
            raise cwe.BadRequest(json.dumps({
                'error': 'Must use floating-point values as values'}))
        if   value1 is None: channel = 2
        elif value2 is None: channel = 1
        else:                channel = None
        values = (value1, value2)
        if channel is not None: values = values[channel - 1]
        return request(self.__thread_state, 'set_' + self._trid_suffix,
                       channel, values=values, **self._trid_kwargs)
```

Листинг 9 — Общий класс-предок для большей части API, взаимодействующей с ТРИД

В соответствии с API **circuits** метод **POST** указанного класса вызывается при POST-запросе по пути, указанном в атрибуте **channel**. Метод **get**, как и любой другой метод, начинающийся с маленькой буквы и содержащий только маленькие буквы и знаки подчёркивания[1] вызывается при GET-запросе по адресу **{channel}/{method_name}**, где **{channel}** — путь, содержащийся в атрибуте **channel**, а **{method_name}** — имя мето-

[1] Всё только из таблицы ASCII символов.

да (в данном случае — **get**). Если всё в порядке, то оба рассматриваемых метода возвращают строку, см. листинг 11. В случае ошибок возбуждаются исключения из модуля **circuits.web.exceptions**, преобразующиеся в соответствующие HTTP коды состояния[23, секция 6].

```
class PIDCoefKpGetSetAPI(TRIDGetSetAPI):
    '''/api/coef/kp/* API entry point

    :param PIDControllerThreadState thread_state:
        PID controller thread state.'''
    channel = '/api/coef/kp'
    _trid_suffix = 'pid_coef'
    _trid_kwargs = {'coef': 'Kp'}
```

Листинг 10 — Пример класса-потомка **TRIDGetSetAPI**

Помимо атрибута **channel** ожидается, что классы-потомки **TRIDGetSetAPI** установят атрибуты **_trid_suffix** и **_trid_kwargs**. Первый атрибут — строка, указывающая метод **trid.TRID** для запроса в функции **request** (используются два метода: **get_{_trid_suffix}** и **set_{_trid_suffix}** для **get** и **POST** соответственно); второй описывает дополнительные аргументы для запрашиваемого метода. Пример класса-потомка приведён в листинге 10, код всех используемых таких классов приведён в листинге 38.

```
import json
import threading
import circuits.web.exceptions as cwe

REQUEST_WAIT_TIMEOUT = 10

def request(thread_state, meth, channel=None, **kwargs):
    '''Request register value from the PID regulator

    :param str meth: :py:class:`trid.TRID` method name.
    :param channel: Channel: 1, 2 or ``None``.
    :param kwargs: Additional arguments, if needed.

    :return: Request body. May raise exceptions from
        :py:module:`circuits.web.exceptions`.'''
    finish_evt = threading.Event()
    kw = {'channel': channel if channel is None else int(channel)}
    kw.update(kwargs)
    response = []
    thread_state.job_queue.put((finish_evt, meth, kw, response))
    if finish_evt.wait(REQUEST_WAIT_TIMEOUT):
        err, value = response[0]
        if not err:
            raise cwe.InternalServerError(*value)
        if value is not None:
            if channel is None:
                value = {'value1': value[0], 'value2': value[1]}
            else:
                value = {'value' + str(channel): value}
        return json.dumps(value)
    else:
        raise cwe.RequestTimeout()
```

Листинг 11 — Вспомогательная функция **request**

В листинге 11 показано, как происходит общение с потоком взаимодействия с ТРИД: в очередь событий помещается событие, создаваемое на месте и затем устанавливаемое в потоке взаимодействия по завершению обработки запроса, имя метода класса **trid.TRID**, аргументы этого метода и контейнер для возврата результата. Ожидается, что после установки события в контейнере окажется пара с двумя значениями: флаг ошибки (ложен, если ошибка, истинен в её отсутсвие) и собственно ответ — описание ошибки в виде списка аргументов для **circuits.web.exceptions.InternalServerError** либо зна-

чение, возвращённое запрошенным методом **trid.TRID**. Возвращённое значение затем преобразуется в строку формата JSON[24]. Для того, чтобы предотвратить бесконечное ожидание в случае программной ошибки ожидание установки события ведётся не более **REQUEST_WAIT_TIMEOUT** (т.е. десяти) секунд, затем возбуждается исключение. Для упрощения кода и вследствие того, что известно, что методы **trid.TRID.set_*** возвращают исключительно **None**, соглашение «POST методы должны вернуть **null** при отсутствии ошибки», прослеживаемое в табл. В.1 не обрабатывается отдельно: **json.dumps(None)** возвращает **null**.

Вследствие наличия состояния «нагрев включён/отключён», а также того, что целевая температура в сервисе является настройкой потока взаимодействия с ТРИД, а не параметром, запрашиваемым у ТРИД, несмотря на внешную схожесть методов для **/api/target** не используется наследование от **TRIDGetSetAPI**: код класса **TargetGetSetAPI**, обрабатывающего приведён в листинге 12.

```
import circuits.web.exceptions as cwe
from circuits.web import Controller

DISABLED_TEMP = -200

class TargetGetSetAPI(Controller):
    channel = '/api/target'
    def __init__(self, thread_state, *args, **kwargs):
        super(TargetGetSetAPI, self).__init__(*args, **kwargs)
        self.__thread_state = thread_state
    def get(self, channel=None):
        if channel not in {None, '1', '2'}:
            raise cwe.BadRequest(json.dumps(
                {'error': 'Unexpected channel'}))
        ret = {}
        with self.__thread_state.state_lock:
            ret['enabled1'], ret['enabled2'] = (
                self.__thread_state.target_enabled)
        if ret['enabled1'] or ret['enabled2']:
            request(self.__thread_state, 'get_target_temp', channel)
        with self.__thread_state.state_lock:
            ret['value1'], ret['value2'] = (
                self.__thread_state.target_temp)
        return json.dumps(ret)
    def POST(self, value1=None, value2=None, enabled1=None, enabled2=None):
        if (value1 is None and value2 is None
            and enabled1 is None and enabled2 is None):
            raise cwe.BadRequest(json.dumps({
                'error': 'Must supply at least one value'}))
```

```
30          try:
31              if value1 is not None: value1 = float(value1)
32              if value2 is not None: value2 = float(value2)
33          except ValueError:
34              raise cwe.BadRequest(json.dumps({
35                  'error': 'Must use floating-point values as values'}))
36          try:
37              if enabled1 is not None: enabled1 = to_boolean(enabled1)
38              if enabled2 is not None: enabled2 = to_boolean(enabled2)
39          except ValueError:
40              raise cwe.BadRequest(json.dumps({
41                  'error': 'Must use boolean values as enabled switches'}))
42          with self.__thread_state.state_lock:
43              target_temp = list(self.__thread_state.target_temp)
44              if value1 is not None: target_temp[0] = value1
45              if value2 is not None: target_temp[1] = value2
46              enabled = list(self.__thread_state.target_enabled)
47              if enabled1 is not None: enabled[0] = enabled1
48              if enabled2 is not None: enabled[1] = enabled2
49              self.__thread_state.set_target_temp(target_temp, enabled)
50          return 'null'
```

Листинг 12 — Код класса **TargetGetSetAPI**, обрабатывающего запросы к **/api/target**

Как видно на листинге, запрос текущей целевой температуры обрабатывается путём получения настроек (атрибутов **PIDControllerThreadState**) **target_temp** (значения температуры) и **target_enabled** (состояния флага отключённости нагрева), защищённого блокировкой **state_lock**. Установка целевой температуры осуществляется путём установки этих же атрибутов и ещё двух под той же блокировкой путём вызова метода **set_target_temp**. Два дополнительных атрибута — это **did_set_target_temp** для указания потоку взаимодействия с ТРИД на необходимость установки целевой температуры у ТРИД и **need_save_settings** для указания ему же на необходимость сохранения изменённых настроек (см. листинг 8).

Получение сохранённых показаний температурных датчиков происходит схожим с получением текущей целевой температуры образом, только вместо **state_lock** используется совместная блокировка **data_lock**. В потоке взаимодействия с ТРИД эта же блокировка используется в исключительном режиме. Для отбрасывания старых показаний (т.е. полученных до временной отметки из параметра **timestamp**) используется модуль **bisect** из стандартной библиотеки Python, позволяющий в одну строку осуществлять поиск методом дихотомии. Код соответствующего класса приведён в листинге 39. Обработка сброса показаний осуществляется отдельно в классе **TemperatureClearAPI** (см. листинг 40).

Помимо этого API сервиса также предоставляет доступ к настройкам подключения ТРИД и настройкам опроса температурных датчиков: **/api/trid/connect** и **/api/trid/monitor**. Код классов-обработчиков данных запросов приведён в листингах 41 и 42 соответственно.

3.6.2 Основная точка входа в сервис

Как видно из пункта 3.6.1 все классы-обработчики API запросов для инициализации требуют параметр **thread_state**. Помимо этого есть желание сделать так, чтобы код запуска сервиса был бы так же прост, как и в листинге 2. Для этого все классы, описанные в пункте 3.6.1 инициализируются в одном классе-точке входа **API**, приведённом в листинге 13.

```
from circuits.web import Controller
class API(Controller):
    '''API entry point, serves nginx /api requests'''
    channel = '/api'
    '''Specify that this works with /api.'''

    def __init__(self, *args, **kwargs):
        super(API, self).__init__(*args, **kwargs)
        self.__thread_state = PIDControllerThreadState()
        self.__thread = PIDControllerThread(self.__thread_state)
        self.__thread.start()
        self += TemperatureAPI(self.__thread_state)
        self += TRIDConnectAPI(self.__thread_state)
        self += TargetGetSetAPI(self.__thread_state)
        self += HysteresisGetSetAPI(self.__thread_state)
        self += AlarmAGetSetAPI(self.__thread_state)
        self += PIDCoefKpGetSetAPI(self.__thread_state)
        self += PIDCoefKiGetSetAPI(self.__thread_state)
        self += PIDCoefKdGetSetAPI(self.__thread_state)
        self += TRIDMonitorAPI(self.__thread_state)
```

Листинг 13 — Класс-точка входа в сервис

Указанный класс является ответственным за инициализацию всех используемых классов-обработчиков запросов, а также запуск потока взаимодействия с ТРИД с инициализацией состояния потока. С его использованием последняя часть приложения (начиная со строки **if __name__ == '__main__'**) выглядит точно так же, как и в листинге 2.

3.7 Web-интерфейс

Описанный выше интерфейс (API) пригоден для использования программами, но он не предназначен для взаимодействия с человеком. В соответствии с пунктом 2.1 для выполнения данной задачи используется nginx, настроенный на раздачу статических файлов, предполагающих использование браузера. Всего для создания Web-интерфейса потребовалось четыре файла:

а) Описание страницы, отображаемой браузером на языке разметки HTML. Размещается в **html/trid.html** и является основной точкой входа.

б) Стилевой файл в формате CSS. Размещается в **html/static/style.css**, управляет отображением страницы.

в) Описание поведения страницы в формате JavaScript. Размещается в **html/static/main.js**.

г) Библиотека dygraph.js (минифицированная и с встроенными в файл зависимостями).

3.7.1 Особенности HTML-кода страницы

Основная и единственная страница Web-интерфейса содержит простейшую иерархию HTML5-тегов, в основном состоящую из тегов **div**, **input** и **form**. Внешний вид страницы показан на рис. 3.2 и 3.3.

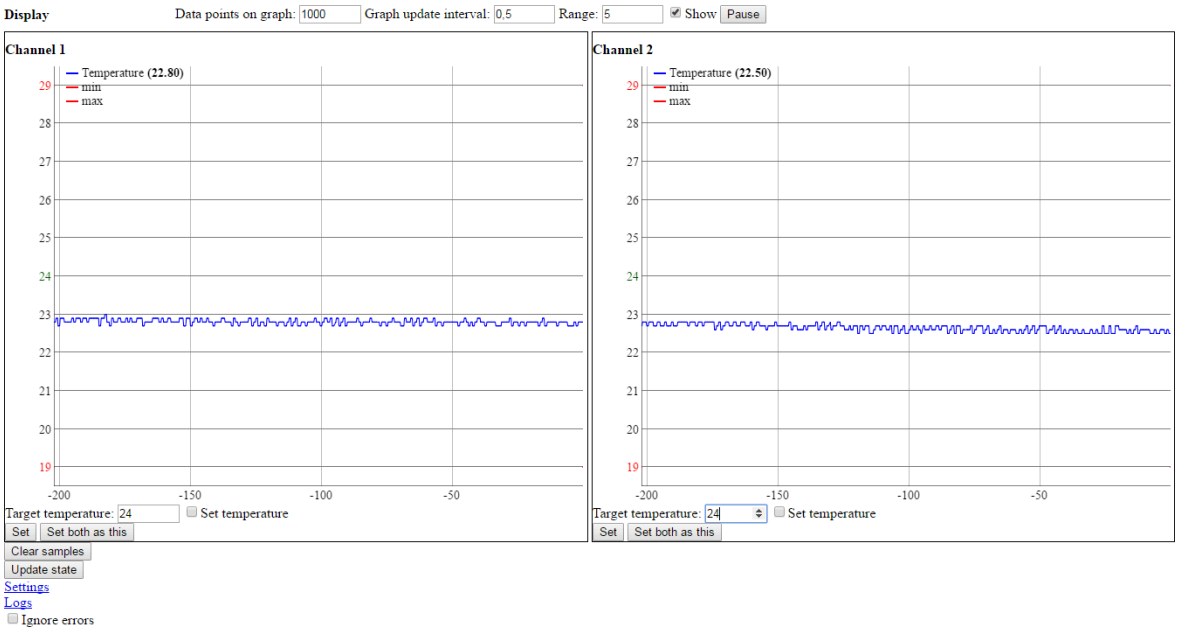


Рисунок 3.2 — Web-интерфейс, основной вид

Как видно из рис. 3.2 и 3.3, страница имеет два основных режима отображения: основной вид, где видны только собственные настройки Web-интерфейса, несколько ссы-

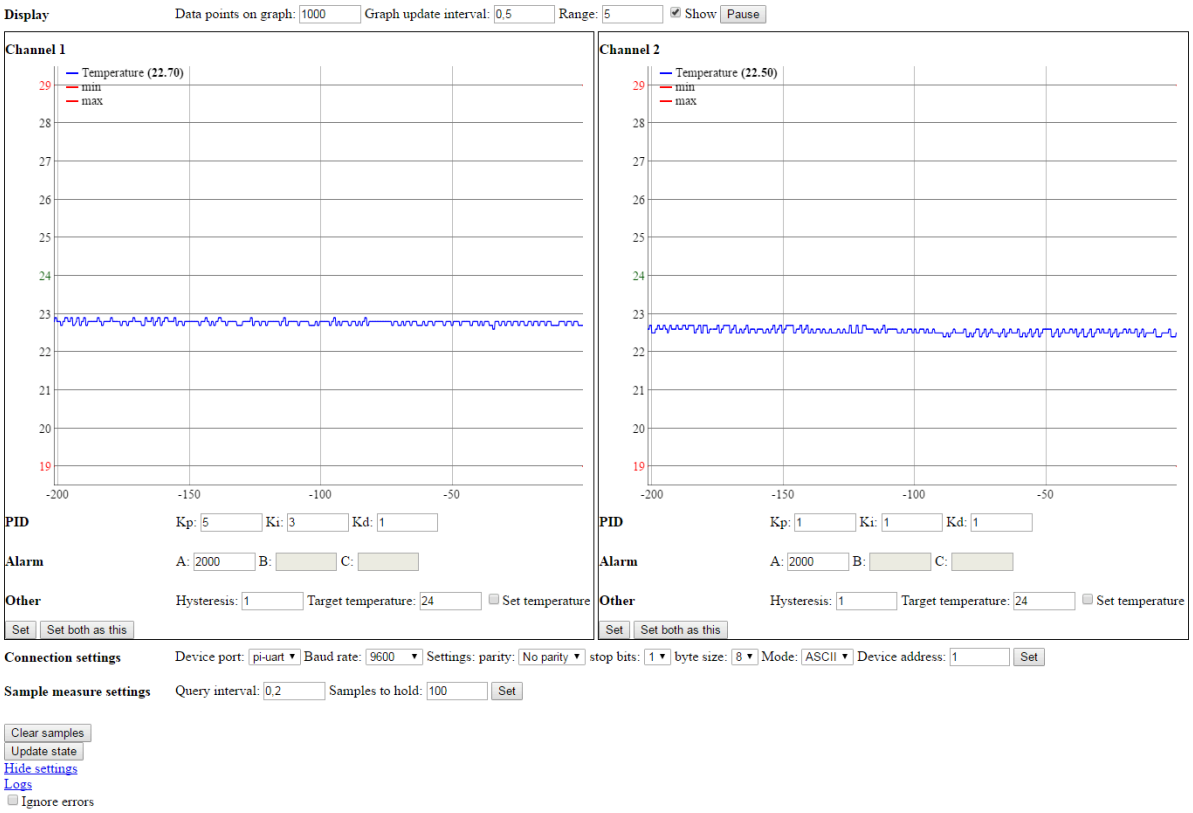


Рисунок 3.3 — Web-интерфейс, вид со всеми доступными настройками

лок на дополнительные режимы и установка целевой температуры, и расширенный вид со всеми параметрами ТРИД, доступными для установки через его RS485 интерфейс. Данное разделение было сделано, чтобы не отвлекать внимание оператора на настройки, изменение которых, скорее всего, не понадобится в ходе эксперимента.

Смена режимов осуществляется путём изменения видимости элементов, а именно манипуляциями CSS атрибутом **display** из JavaScript кода. В связи с тем, что осуществление данных манипуляций напрямую довольно неудобно, JavaScript изменяет атрибут **display** путём присвоения или удаления CSS класса у выбранных тегов **div**.

В HTML коде также определены реакции на пользовательский ввод: а именно

а) Изменения собственных настроек Web-интерфейса применяются немедленно. Дополнительно при нажатии кнопки «ввод» на клавиатуре при поле настройки (если она представляет собой поле для ввода) в фокусе также происходит применение настройки, на случай если событие «изменение поля» не работает. Пример определения поля с собственной настройкой представлен в листинге 14. Пример определения булевой собственной настройки Web-интерфейса представлен в листинге 15.

б) Изменения параметров ТРИД применяются после нажатия кнопки «ввод» на клавиатуре при соответствующем поле в фокусе, либо после использования кнопок «Set» или «Set both as this». Код всех таких параметров для одного канала представлен в листинге 16.

в) Аналогично для настроек параметров подключения к ТРИД и параметров опроса ТРИД. HTML-код, однако имеет несколько дополнительных особенностей и потому вынесен отдельно в листинги 17 (параметры подключения) и 18 (параметры опроса).

```
<form
  class="control"
  onsubmit="return window.trid.changed_graph_data_points()">
    Data points on graph: <input
      type="number"
      id="graph-max-data_points"
      step="any"
      min="0"
      class="num-input"
      onchange="return window.trid.changed_graph_data_points()"/>
</form>
```

Листинг 14 — Пример собственной настройки Web-интерфейса, поле для ввода

```
<input
  id="ignore-errors-1"
  type="checkbox"
  onchange="return window.trid.toggle_ignore_errors(this)"
/><label>Ignore errors</label>
```

Листинг 15 — Пример собственной настройки Web-интерфейса, булево значение

```
<div class="advanced">
    <h2 class="control-header">PID</h2>
    <form
      id="trid-state-coef-kp-value1-form"
      onsubmit="return window.trid.handle_event(this)"
      class="control">
        Kp: <input
          id="trid-state-coef-kp-value1-input"
          type="number" step="0.1" min="0" max="3000"
          class="num-input"/>
    </form>
    <form
      id="trid-state-coef-ki-value1-form"
      onsubmit="return window.trid.handle_event(this)"
      class="control">
```

```
      Ki: <input
        id="trid-state-coef-ki-value1-input"
        type="number" step="1" min="0" max="9999"
        class="num-input"/>
    </form>
    <form
      id="trid-state-coef-kd-value1-form"
      onsubmit="return window.trid.handle_event(this)"
      class="control">
      Kd: <input
        id="trid-state-coef-kd-value1-input"
        type="number" step="0.1" min="0" max="999.9"
        class="num-input"/>
    </form>
    <br/>
    <h2 class="control-header">Alarm</h2>
    <form
      id="trid-state-alarm-a-value1-form"
      onsubmit="return window.trid.handle_event(this)"
      class="control">
      A: <input
        id="trid-state-alarm-a-value1-input"
        type="number" step="0.1" min="-200" max="2500"
        class="num-input"/>
    </form>
    <form
      id="trid-state-alarm-b-value1-form"
      onsubmit="return window.trid.handle_event(this)"
      class="control">
      B: <input
        id="trid-state-alarm-b-value1-input"
        type="number" step="0.1" min="-200" max="2500"
        class="num-input"
        disabled="1"/>
    </form>
    <form
      id="trid-state-alarm-c-value1-form"
      onsubmit="return window.trid.handle_event(this)"
      class="control">
      C: <input
        id="trid-state-alarm-c-value1-input"
        type="number" step="0.1" min="-200" max="2500"
        class="num-input"
        disabled="1"/>
    </form>
```

```
</div>
<h2 class="control-header advanced">Other</h2>
<span class="advanced">
    <form
      id="trid-state-hyst-value1-form"
      onsubmit="return window.trid.handle_event(this)"
      class="control">
        Hysteresis: <input
          id="trid-state-hyst-value1-input"
          type="number" step="0.1" min="0.1" max="50"
          class="num-input"/>
    </form>
</span>
<form
  id="trid-state-target-value1-form"
  onsubmit="return window.trid.handle_event(this)"
  class="control">
    Target temperature: <input
      id="trid-state-target-value1-input"
      type="number" step="0.1" min="-200" max="2500"
      class="num-input"/>
    <input
      id="trid-state-target-enabled1-input"
      type="checkbox"
      onchange="return window.trid.handle_event(this)"
    /><label>Set temperature</label>
</form>
```

Листинг 16 — Пример кода параметров одного канала

```
<div class="advanced">
    <h1 class="control-header">Connection settings</h1>
    <form
      id="trid-state-trid-connect-all-form"
      onsubmit="return window.trid.handle_event(this)"
      class="control">
        Device port:
        <select id="trid-state-trid-connect-port-input">
            <option value="">Default</option>
        </select>
        Baud rate:
        <select id="trid-state-trid-connect-baudrate-input">
            <option value=""></option>
            <option value="9600">9600</option>
```

```
            <option value="19200">19200</option>
            <option value="28800">28800</option>
            <option value="57600">57600</option>
            <option value="115200">115200</option>
        </select>
        Settings: parity:
        <select id="trid-state-trid-connect-parity-input">
            <option value=""></option>
            <option value="N">No parity</option>
            <option value="O">Odd</option>
            <option value="E">Even</option>
        </select>
        stop bits:
        <select id="trid-state-trid-connect-stopbits-input">
            <option value=""></option>
            <option value="1">1</option>
            <option value="2">2</option>
        </select>
        byte size:
        <select id="trid-state-trid-connect-bytesize-input">
            <option value=""></option>
            <option value="8">8</option>
            <option value="7">7</option>
        </select>
        Mode:
        <select id="trid-state-trid-connect-mode-input">
            <option value=""></option>
            <option value="asc">ASCII</option>
            <option value="rtu">RTU</option>
        </select>
        Device address:
        <input
          id="trid-state-trid-connect-slaveaddress-input"
          class="num-input"
          type="number" step="1" min="1" max="255"/>
        <input
          type="submit"
          id="trid-state-trid-connect-all-submit"
          onclick="return window.trid.handle_event(this)"
          value="Set"/>
    </form>
</div>
```

Листинг 17 — Код параметров подключения к ТРИД

```
<div class="advanced">
    <h1 class="control-header">Sample measure settings</h1>
    <form
      id="trid-state-trid-monitor-all-form"
      onsubmit="return window.trid.handle_event(this)"
      class="control">
        Query interval:
        <input
          id="trid-state-trid-monitor-interval-input"
          type="number" step="0.1" min="0" max="60"
          class="num-input"/>
        Samples to hold:
        <input
          id="trid-state-trid-monitor-data_points-input"
          type="number" step="1" min="2"
          class="num-input"/>
        <input
          type="submit"
          id="trid-state-trid-monitor-all-submit"
          onclick="return window.trid.handle_event(this)"
          value="Set"/>
    </form>
</div>
```

Листинг 18 — Код параметров опроса ТРИД

Особенности HTML-кода параметров ТРИД, подключения к ТРИД и опроса ТРИД:

а) Идентификаторы (атрибут **id**) форм, полей ввода и переключателей указаны таким образом, чтобы JavaScript код имел возможность соотнести их с API запросами. К примеру для установки коэффициента Ki канала 2 нужно отправить POST запрос методу **/api/coef/ki** с параметром **value2**. Соответствующее поле ввода таким образом называется **trid-state-coef-ki-value2-input**, а форма (тег **form**) — **trid-state-coef-ki-value2-form**.

б) Все поля и формы используют одну JavaScript функцию для обработки событий — **handle_event**, которая получает единственный аргумент **this**. Причина: при создании событий браузер исполняет JavaScript код в атрибуте **on…** (к примеру, **onclick**) с **this** установленным в DOM-элемент тега, вызвавшего события. Вместе с описанной выше схемой наименования данный подход позволяет не увеличивать число функций-обработчиков событий.

в) Вместо одной формы на все поля ввода сразу используется множество форм, на каждое поле ввода по отдельности. Данная особенность позволяет точно определять, в каком поле ввода пользователь нажал «ввод», таким образом избегая ненужных вычислений и ненужных API вызовов. (Это не относится к параметрам подключения и опроса, т.к. параметры подключения устанавливаются одним API вызовом, также как и оба параметра опроса.)

г) HTML-код содержит граничные значения из табл. В.4. Причина была рассмотрена в пункте 3.4: превышение/принижение граничных значений не вызывает отказа ТРИД, поэтому их лучше сохранить в наиболее легко изменяемой части ПО.

д) При параметре «Device port» отсутствует список портов: это единственное списковое значение, получаемое через API.

е) Элементы с классом **advanced** отображаются только в режиме с показом расширенных настроек. Для этого они должны находится внутри **div** с CSS классом **advanced-disabled** и идентификатором **advanced-disabler** (последнее нужно для отключения режима расширенных настроек и перехода к основному режиму).

В обоих режимах присутствуют графики, однако для показа графика с выбранной библиотекой в HTML коде достаточно просто выделить под график **div**, который можно увидеть в листинге 19. Всю дальнейшую работу берёт на себя библиотека dygraph.js.

```
<div class="ch-block">
    <h1 class="control-header">Channel 1</h1>
    <br/>
    <div id="graph-channel-1" class="graph"></div>
    <!-- Код параметров канала -->
</div>
```

Листинг 19 — Блок канала без параметров

Помимо рассмотренных выше режимов существуют ещё два: режим показа журнала и режим показа ошибки API. Первый представляет собой просто чёрный **div** на весь экран с белым текстом журнала и кнопками закрытия (одна в самом начале, другая в самом конце) (см. листинг 20). Второй — частично прозрачный чёрный **div** на весь экран (затеняет основную часть интерфейса) с сообщением об ошибке на белом фоне, кнопкой закрытия сообщения и переключателем, позволяющим прекратить показ ошибок (см. листинг 21). Предоставлением текста журнала или ошибки занимается JavaScript, HTML только содержит контейнеры, тогда как CSS определяет их отображение.

```
<div id="logs" class="logs-disabled">
    <input
      type="button"
      onclick="return window.trid.hide_logs()"
      value="Close"/>
    <div id="logs-container" class="logs-container">
    </div>
    <div id="logs-progress" class="logs-progress">
    </div>
    <input
      type="button"
      class="error-close-button"
      onclick="return window.trid.hide_logs()"
      value="Close"/>
</div>
```

Листинг 20 — HTML-код режима отображения журнала

```
<div id="error" class="error-disabled">
    <div class="error-message-container">
        <div id="error-message" class="error-message">
        </div>
        <div class="error-controls">
            <input
              type="button" class="error-close-button"
              onclick="return window.trid.hide_error()"
              value="Close"/>
            <br/>
            <input
              id="ignore-errors-1"
              type="checkbox"
              onchange="return window.trid.toggle_ignore_errors(this)"
            /><label>Ignore errors</label>
        </div>
    </div>
</div>
```

Листинг 21 — HTML-код режима отображения ошибки

3.7.2 Собственные настройки Web-интерфейса

Как видно из рис. 3.2, Web-интерфейс имеет пять собственных настроек: четыре наверху страницы, одна внизу. Первые отличаются тем, что JavaScript хранит установ-

ленные пользователем значения указанных настроек в **window.localStorage**, последняя же хранится только до закрытия либо обновления страницы: скрываемые ей ошибки являются исключительными ситуациями и их замалчивание может привести к тому, что команды оператора будут игнорироваться, но оператор не будет осведомлён об этом.

Хранение настроек в **window.localStorage** означает, что при обновлении страницы Web-интерфейса и в целом при использовании одного и того же компьютера с одним и тем же браузером для работы с Web-интерфейсом следующая сессия продолжит использовать настройки, использованные в предыдущей сессии. Однако **window.localStorage** может быть очищен пользователем при необходимости, так же при использовании разных компьютеров или разных браузеров на одном компьютере настройки не сохраняются.

Список всех настроек:

«Ignore errors» — обычно Web-интерфейс отображает ответы на API запросы, HTTP код которых отличен от 200[23, секция 6], как ошибки. При включении данной настройки такие ответы будут игнорироваться. Настройка не сохраняется в **window.localStorage**.

«Data points on graph» — количество хранимых в Web-интерфейсе показаний температуры для отображения на графике. Может быть как больше, так и меньше количества хранимых показаний в API backend. В случае бóльшего количества хранимых показаний по сравнению с API backend дополнительные точки исчезают после обновления страницы Web-интерфейса.

«Graph update interval» — максимальная частота запроса новых показаний температуры и, соответственно, обновления графика. Реальная частота запроса может отличаться от заданной из-за задержек, вносимых сетью, и общей нестабильности интервалов: JavaScript в браузере является однопоточным, поэтому обработка команд оператора может задержать обновление графика, кроме того системный планировщик задач в бóльшинстве используемых ОС периодически меняет текущую исполняемую процессом задачу, одной из которых является браузер.

«Range» — как видно на рис. 3.2, на графике отображаются три выделенных цветом значения. Одно, выделенное зелёным, соответствует параметру «target temperature» внизу. Два других, выделенных красным, соответствуют данному параметру, к которому прибавили или из которого вычли «range». Эта настройка влияет только на внешний вид Web-интерфейса, её изменение больше ни на что не влияет. На рис. 3.4 представлен внешний вид Web-интерфейса с отключённым показом диапозона.

«Show» — при отключении данной настройки с графика пропадают выделенные цветом элементы сетки. Так же, как и с «range», изменение данной настройки влияет только на отображение.

HTML код полей ввода всех описанные выше настроек, кроме «Ignore errors» приведён в листинге 43. Код одного из флажков, соответствующих настройке «Ignore errors» есть в листинге 21.

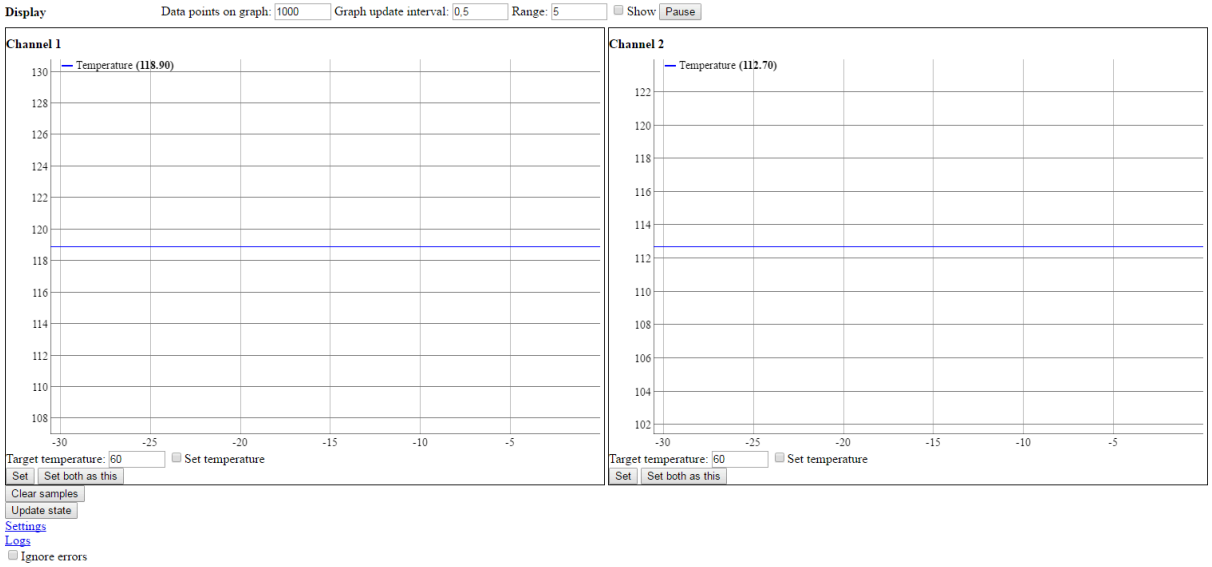


Рисунок 3.4 — Внешний вид Web-интерфейса с отключённым показом диапозона

3.7.3 Описание CSS кода страницы

Использованные CSS классы представлены в листинге 22. Далее приводится их описание:

Класс **control** используется для тегов **form**, позволяя размещать две формы в одной строчке без привлечения дополнительных контейнеров.

Класс **num-input** управляет отображением полей ввода, используемых для ввода чисел. Благодаря ему все поля ввода чисел имеют одинаковую ширину.

Класс **control-header** используется для жирных подписей около строчек с параметрами или настройками: «Display», «PID», «Alarm», «Other», «Connection settings», «Sample measure settings». Позволяет размещать следующие элементы, с одной стороны, в одной линии с подписью, с другой стороны, точно друг под другом.

Класс **ch-block** применяется для контейнера, содержащего всё, связанное с одним каналом (т.е. график и параметры, а также заголовок вида «Channel 1»). Применяется для рисования единственных двух рамок.

Класс **advanced** применяется для скрытия определённых параметров в обычном режиме и отображения их в режиме с расширенными настройками. Скрытие происходит, если элемент с данным классом попадает внутрь контейнера с классом **advanced-disabler**.

Класс **graph** задаёт размеры графика: 48 % от экрана по ширине, 50 % по высоте. Два процента по ширине зарезервированы под рамки.

Класс **error** задаёт отображение **div**'а, используемого для затенения интерфейса при отображении ошибки: данный **div** должен закрывать весь интерфейс, затеняя его.

Класс **error-disabled** используется для скрытия ошибки и указанного выше **div**'а.

Класс **error-message-container** используется для контейнера, в котором отображается текст ошибки. Он обеспечивает белый непрозрачный фон и отсутвие затенения, а также сдвиг от верхнего края экрана.

Класс **error-controls** центрирует кнопку закрытия текста ошибки и настройку «Ignore errors».

Класс **error-message** управляет отображением контейнера, непосредственно содержащего текст ошибки.

Класс **logs-disabled** используется для скрытия **div**'а режима просмотра журнала.

Класс **logs** используется для отображения **div**'а режима просмотра журнала. Указанный **div** данным классом растягивается на весь экран и закрывает все элементы интерфейса, не относящиеся к режиму просмотра журнала. Класс назначается из JavaScript тому же элементу, для скрытия которого в нормальном режиме используется класс **logs-disabled**.

Класс **logs-progress** используется для отображения индикатора прогресса загрузки журнала.

Класс **dygraph-legend** нужен для того, чтобы фон легенды на графике был прозрачным, а сама легенда располагалась неподалёку от оси слева.

```
.control {
    display: inline;
}
.num-input {
    max-width: 5em;
}
.control-header {
    display: inline-flex;
    font-size: inherit;
    width: 12em;
}
.ch-block {
    display: inline-block;
    border-color: black;
    border-width: 1px;
    border-style: solid;
}
.advanced-disabled .advanced {
    display: none;
}
```

```
.graph {
    width: 48vw;
    height: 50vh;
}
.error {
    position: absolute;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    z-index: 100;
    background-color: rgba(0, 0, 0, 0.2);
}
.error-disabled {
    display: none;
}
.error-message-container {
    position: relative;
    top: 10%;
    z-index: 101;
    background-color: rgba(255, 255, 255, 1);
}
.error-controls {
    text-align: center;
}
.error-message {
    display: block-inline;
}
.logs-disabled {
    display: none;
}
.logs {
    position: absolute;
    left: 0;
    top: 0;
    min-width: 100%;
    min-height: 100%;
    z-index: 50;
    background-color: rgba(0, 0, 0, 1);
    color: rgba(255, 255, 255, 1);
}
.logs-progress {
    color: red;
}
.dygraph-legend {
```

```
66      background-color: transparent !important;
67      left: 5em !important;
68  }
```

Листинг 22 — Код **style.css**

3.7.4 Описание JavaScript кода страницы

Функции, определённые в JavaScript коде страницы, можно условно разделить на следующие группы:

а) Функции работы с настройками Web-интерфейса.
б) Функции работы с графиком.
в) Функции работы с параметрами Web-сервиса.
г) Функции режима показа журнала.
д) Функции режима показа ошибки.
е) Обработчики событий.

Все указанные группы находятся внутри одной большой функции, вызываемой при событии **onload**, т.е. при окончании загрузки страницы (см. листинг 23). Данная особенность нужна для создания собственной области видимости JavaScript кода — практика, направленная на уменьшение числа потенциальных проблем при использовании множества JavaScript файлов, — а также для того, чтобы запуск JavaScript кода не провалился из-за незагруженных ресурсов.

```
1  window.onload = (function () {
2      // Весь код здесь
3  });
```

Листинг 23 — Верхий уровень JavaScript кода

3.7.4.1 Функции работы с сохраняемыми настройками Web-интерфейса

Для работы с настройками Web-интерфейса используются три функции, две константы и две переменные состояния. Основная особенность настроек заключается в том, что имеются три источника значения настроек: значения по-умолчанию, значения, сохранённые в **window.localStorage** (нужны для сохранения настроек с предыдущей сессии) и значения в полях ввода. Поэтому схема получения настройки выглядит так:

а) Если установлен флаг «предпочитать значения из поля ввода», просто взять значение из поля ввода. Флаг устанавливается в обработчике события изменения настройки.

б) Иначе получить значение из **window.localStorage**. Если значения там нет (к примеру, не было предыдущей сессии), то использовать умолчания.

После того, как значение настройки получено, значение поля ввода настройки устанавливается в соответствии с полученным значением настройки, за исключением случаев, когда указанное поле ввода находится в фокусе (т.е. редактируется оператором) — внезапное изменение редактируемого значения может помешать оператору. Затем значение настройки сохраняется в **window.localStorage**.

Таким образом, для работы с настройками используются следующие сущности, представленные в листинге 24:

Переменная состояния storage просто хранит ссылку на **window.localStorage** для удобства написания кода.

Функции setStorage и getStorage отвечают за сохранение и получение настроек: они хранятся в **window.localStorage** как есть, сериализованные в формат **JSON** (**window.localStorage** позволяет хранить только строки) с ключом, соответствующим названию настройки.

Константа displayInputs хранит ссылки на поля ввода, соответствующие настройкам.

Константа displayOptionsDefaults содержит настройки по-умолчанию.

Переменная состояния displayPreferInput содержит флаги, устанавливаемые событиями изменения полей ввода настроек. Данные флаги указывают, что при следующем получении настройки нужно использовать не сохранённое в **window.localStorage** значение, а значение из поля ввода.

Использующие данную переменную функции-обработчики событий из листинга 48 устанавливают один или несколько из флагов в **displayPreferInput**, после чего одна из них вызывает обновление графика (при изменении настроек «Range» и «Show»), другая перезапускает таймер, описанный далее в пункте 3.7.4.3 (при изменении настройки «Graph update interval»), третья просто устанавливает флаг (он будёт применён при следующем получении значения настройки).

Функция getDisplayOption отвечает за получение значения настройки в JavaScript коде, а также за его сохранение в **window.localStorage** (таким образом, если появятся неиспользуемые настройки, то они не будут сохраняться!). Именно в ней закодировано описанное в начале данного пункта поведение.

Также отдельно нужно отметить две функции для уницифицированной работы с полями ввода различного типа, использованные в **getDisplayOption**: **getInputValue** и **setInputValue**. Функции предполагают получение значения настройки из полей вво-

```
var storage = window.localStorage;
var setStorage = function(key, val) {
    storage.setItem(key, JSON.stringify(val));
};
var getStorage = function(key) {
    return JSON.parse(storage.getItem(key));
};
var displayInputs = {
    maxDataPoints: document.getElementById('graph-max-data_points'),
    queryInterval: document.getElementById('graph-query-interval'),
    rng: document.getElementById('graph-rng'),
    showRange: document.getElementById('graph-show-rng'),
};
var displayOptionsDefaults = {
    maxDataPoints: 1000,
    queryInterval: 0.5,
    rng: 5,
    showRange: true,
};
var displayPreferInput = {};

var getDisplayOption = function(option) {
    var ret;
    var input = displayInputs[option];
    if (displayPreferInput[option]) {
        ret = getInputValue(input);
        displayPreferInput[option] = false;
    } else {
        ret = getStorage(option);
        if (ret === undefined || ret === null) {
            ret = displayOptionsDefaults[option];
        }
    }
    if (input !== document.activeElement) {
        setInputValue(input, ret);
    }
    setStorage(option, ret);
    return ret;
};
```

Листинг 24 — Код для работы с сохраняемыми настройками Web-интерфейса

да типа «checkbox» (тег **input** с атрибутом **type="checkbox"**: флажок, булевы значения),

«select» (тег **select**: список выбора, строковые значения), «number» (тег **input** с атрибутом **type="number"**: поле ввода чисел, числа с плавающей точкой). На входе в качестве первого аргумента обе функции принимают DOM-элемент поля ввода. Основная задача: спрятать тот факт, что для флажков значения сохранены в атрибуте **checked** DOM-элемента (в остальных случаях используется атрибут **value**), а также привести числовые значения к числу. Код функций представлен в листинге 25.

```
var getInputValue = function(el) {
    return (el.type === 'checkbox' ? el.checked : (
                el.type === 'number' ? +el.value : el.value));
};
var setInputValue = function(el, value) {
    if (el.type === 'checkbox') {
        el.checked = !!value;
    } else {
        el.value = value;
    }
};
```

Листинг 25 — Функции работы с полями ввода

3.7.4.2 Функции работы с несохраняемыми настройками Web-интерфейса

Отдельно от остальных настроек стоит настройка «Ignore errors»: в силу опасности её установки, её значение не сохраняется в **window.localStorage**, оказываясь ложным при каждой новой загрузке Web-интерфейса. Также особенной её делает наличие нескольких флажков для её установки: один доступен из основного режима и из режима с расширенными настройками, другой доступен из режима показа ошибки. Соответственно для её работы потребовалась одна константа и одна переменная состояния, показанные в листинге 26. Весь код для работы с константой и переменной состояния находится в функции обработки события изменения данных флажков и делает всего две вещи: инвертирование переменной состояния (единственная переменная состояния **ignoreErrors** преставляет из себя просто булево значение) и установку флажков в соответствующее новому значению состояние (см. листинг 49).

3.7.4.3 Функции работы с графиком

В JavaScript коде определены следующие функции работы с графиком:

а) Описание графика: функция **getGraphOptions**, вызываемая два раза для определения настроек отображения двух графиков. Фунция служит третьим аргументом при

```
var ignoreErrors = false;
var ignoreErrorsInputs = [
    document.getElementById('ignore-errors-1'),
    document.getElementById('ignore-errors-2'),
];
```

Листинг 26 — Константа и переменная состояния для настройки «Ignore errors»

инициализации объекта **Dygraph**, первые два — ссылка на DOM-элемент — контейнер для графика — **div** с ID **graph-channel-N** (см. листинг 19); и начальные данные для графика (массив массивов данных, в данном случае — **[[0, 0, 0, 0]]**).

б) Функции обновления графика: **getSamples**, запускающая и обрабатывающая API запрос; **updateGraph**, вызываемая из **getSamples** для обновления графика на основе обновлённых данных и **queryTRID**, вызываемая с заданным интервалом (см. пункт 3.7.2) для обновления графика путём вызова **updateGraph** при условии, что оно не было приостановлено.

Функция getGraphOptions содержит описание всех деталей отображения графика:

а) Тип графика — не указан явно, используется тип по-умолчанию.

б) Отображение легенды — отображается всегда, в виде «{линия} {описание линии} (последнее значение температуры)». {Линия} представляет собой линию в цвете (на рис. 3.2 их три: температура (синяя), min и max (красные), последние два могут отсутствовать). {Описание линии} — «Temperature», «min» или «max». «Последнее значение температуры» показывается только для линии «temperature», само значение и скобки отображаются полужирным начертанием.

в) Отображение значений на графике при наведении курсора: отображаются с точностью до сотых.

г) Отображение сетки: сетка по X оставлена по-умолчанию, в сетку по Y добавляются при необходимости линии, соответствующие минимуму и максимуму. Также в этой части определены цвета подписей к линиям сетки (красный и зелёный).

д) Цвета графика: синий для температуры, красный для «граничных» значений.

Полный код описания отображения графика (функции **getGraphOptions** и двух её вызовов) приведён в листинге 44.

Функция queryTRID состоит всего из двух строчек: в одной осуществляется проверка флага паузы, установка и удаление которого происходит в обработчике события нажатия кнопки «Pause»/«Resume», и ранний выход в случае его установки, во второй

осуществляется вызов функции **getSamples**. Полный код функции, а также одного из двух мест её использования, приведён в листинге 27.

```
var paused = false;

var queryTRID = function() {
    if (paused) return;
    getSamples(g1, g2, data);
};
var queryInterval = setInterval(
        queryTRID, getDisplayOption('queryInterval') * 1000);
```

Листинг 27 — Код функции **queryTRID** и её использование

Как видно из кода, функция queryTRID вызывается каждые $q \cdot 1\,000$ мс, где q — значения настройки «Graph update interval» (см. пункт 3.7.2). Переменная состояния **queryInterval** нужна для отмены таймера при изменении настройки. После отмены таймера создаётся новый, который сохраняется в той же переменной состояния, код создания полностью идентичен приведённому в листинге 27, за исключением отсутствия **var**.

Функция getSamples получает данные из API вызова **/api/temp/samples_since**, которые затем сохраняются в массиве **data** (переменная состояния). Функция использует встроенное в браузер API **XMLHttpRequest** и содержит код, удаляющий первые несколько более не отображаемых точек из массива **data**, а также подготовлена к ситуации, когда API вызов вернул больше точек, чем нужно отобразить: максимальное количество отображаемых точек определено в настройке Web-интерфейса «data points on graph» (см. пункт 3.7.2). Код функции **getSamples** представлен в листинге 28.

```
var data = [];
var getSamples = function(g1, g2, data) {
    if (!setPending()) {
        return;
    }
    var xhr = new XMLHttpRequest();
    var suf = '';
    if (data.length > 0) {
        var since = data[data.length - 1][0];
        suf = '?timestamp=' + since;
    }
    xhr.open('GET', url('/api/temp/samples_since' + suf), true);
    xhr.onreadystatechange = function() {
        if (xhr.readyState !== 4) return;
        if (xhr.status !== 200) {
```

```
            showError(xhr.responseText, [xhr]);
            clearPending();
            return;
        }
        var response = JSON.parse(xhr.responseText);
        var unneeded = ((data.length + response.length)
                        - getDisplayOption('maxDataPoints'));
        if (unneeded > data.length) {
            data.length = 0;
            if (unneeded - data.length >= response.length) {
                response = [];
            } else {
                response.splice(0, unneeded - data.length);
            }
        } else if (unneeded > 0) {
            data.splice(0, unneeded);
        }
        for (var i = 0 ; i < response.length ; i++) {
            var el = response[i];
            data.push([(el[0]), el[1][0], el[1][1]]);
        }
        updateGraph(g1, g2, data);
        clearPending();
    };
    xhr.send();
};
```

Листинг 28 — Код функции **getSamples**

Здесь для предотвращения параллельной отправки нескольких запросов используется пара функций **setPending**/**clearPending**. Первая функция проверяет флаг **pendingGet** (переменная состояния) и возвращает ложь, если он установлен, что означает, что предыдущий запрос ещё не завершился. Если он не установлен, то она устанавливает этот флаг и создаёт однократно исполняемый таймер, который должен снять флаг через десять секунд независимо от того, завершился ли вызов API, после чего возвращает истину. Вторая функция убирает флаг и отменяет таймер. Для отмены таймера используется переменная состояния **pendingTimeout**. В отличие от альтернативного решения — запрос API выполняется в однократно исполняемом таймере, который переустанавливается при получении ответа, — решение с флагом позволяет интерфейсу продолжить функционировать, если ответ на предыдущий запрос не был получен в разумное время. Код функций представлен в листинге 29.

```
var pendingGet = false;
var pendingTimeout = null;

var clearPending = function() {
    pendingGet = false;
    clearTimeout(pendingTimeout);
    pendingTimeout = null;
};
var setPending = function() {
    if (pendingGet) {
        return false;
    }
    var pendingTimeoutTime = 10000;
    pendingGet = true;
    pendingTimeout = setTimeout(clearPending, pendingTimeoutTime);
    return true;
};
```

Листинг 29 — Код функций **setPending** и **clearPending**

Также необходимо отметить функцию **url**, используемую на пути API запроса. Данная функция была введена для обеспечения возможности работы API backend на отдельном сервере или же без использования nginx в качестве обратного прокси. В связи с тем, что ни то, ни то в данный момент не используется, функция просто возвращает свой аргумент неизменённым: см. листинг 30.

```
var url = function(u) {
    return u;
};
```

Листинг 30 — Код функции **url**

3.7.4.4 Функции работы с параметрами Web-сервиса

Для всех параметров Web-сервиса (параметров ТРИД: коэффициентов ПИД-регулирования, целевой температуры, …; параметров подключения к ТРИД; параметров опроса ТРИД) был введён единый механизм их отображения и изменения. В рамках этого механизма все параметры сохраняются в словаре состояния. Кроме того, есть взаимосвязь между различными наименованиями и методами получения/установки парамет-

ров: в качестве примера возьмём параметр **port**, определяющий порт подключения ТРИД к Raspberry Pi:

а) Для получения значения данного параметра нужно отправить GET API запрос по адресу **/api/trid/connect/get** (см. табл. В.1). Данный запрос вернёт словарь в котором одним из ключей будет **port**; значение, соответствующее данному ключу будет значением искомого параметра.

б) Для установки значения данного параметра нужно отправить POST API запрос по адресу **/api/trid/connect** (см. табл. В.1) с аргументом **port**.

в) В словаре состояния последнее полученное значение данного параметра сохраняется как значение, соответствующее ключу **port** словаря **connect**, сохранённого внутри словаря **trid**, сохранённого внутри собственно словаря состояния: **{trid: {connect: {port: "pi-uart"}}}**. Т.е. имея словарь состояния **state**, получить нужный параметр можно при помощи следующего кода: **state.trid.connect.port** (с другим синтаксисом: **state["trid"]["connect"]["port"]**).

г) В HTML коде данному параметру соответствует **input** тег с атрибутом **id="trid-state-trid-connect-port-input"**, находящийся внутри формы с атрибутом **id="trid-state-trid-connect-all-form"** (см. пункт 3.7.1).

Как видно из списка, для данного параметра можно выделить следующие сущности:

а) Путь (**objPath**) — часть API запроса без **/api/**, разбитая по косой черте: **["trid","connect"]**.

б) Имя параметра (**key**) — аргумент для POST API запроса: **"port"**.

в) Уникальный идентификатор параметра — пара из пути и имени параметра: **[["trid","connect"], "port"]**.

При этом

а) Для получения значения некоторого параметра нужно отправить GET API запрос по адресу **"/api/"+ objPath.join("/") + "/get"** и взять значение **key** из полученного словаря.

б) Для установки значения некоторого параметра нужно отправить POST API запрос по адресу **"/api/"+ objPath.join("/")** с аргументом **key** равным новому значению.

в) Для сохранения полученного значения в словаре состояния **state** нужно использовать код

```
var getObjAtPath = function(state, path) {
    var obj = state;
    for (var i = 0; i < path.length; i++) {
```

```
                    obj = obj[path[i]];
                }
                return obj;
            };
            var obj = getObjAtPath(state, objPath);
            obj[key] = value;
```

Листинг 31 — Код сохранения значения параметра в словаре

г) В HTML коде параметру соответствует тег с идентификатором (атрибутом **id**) **"trid-state-"+ objPath.join("-") + "-"+ key + "-input"**. С идентификатором формы, однако, всё не так однозначно, т.к. большинство параметров не сгруппированы по формам так, как параметры подключения к ТРИД.

Механизм работы с параметрами описывается следующим образом:

а) При загрузке Web-интерфейса переменная состояния **state** инициализируется копией начального режима (сохранённого в постоянной **initialState**), переменная состояния **previousState** инициализируется копией **state**. Код инициализации приведён в листинге 32.

Для копирования используется функция **deepCopy**, создающая полную копию словаря, включая вложенные словари. Копия создаётся путём сериализации словаря в формат JSON и десериализации из него, таким образом вся функция занимает всего одну строчку (но не способна скопировать ряд значений, которые могут быть помещены в словарь: к примеру, нельзя скопировать функции). Код **deepCopy** приведён в листинге 33.

б) Для отображения текущих значений параметров Web-сервиса необходимо знать, какие параметры отображены в данном режиме. Для этого были заведены две константы **AERegular** и **AEAdvanced** и одна переменная состояния **AE**. Все перечесленные константы и переменные являются словарями с тремя ключами: **endpoints** с массивом пар (массив имён параметров; путь); **channelEndpoints** с таким же массивом (при этом массив внутри **channelEndpoints** может содержать только пары из массива **endpoints**); и **nonChannelEndpoints** со словарём, в котором ключами являются последние части пути, а значениями — те же пары из массива **endpoints**, но не вошедшие в массив **channelEndpoints**.

Здесь в массиве **endpoints** перечислены все отображаемые в данном режиме параметры. В массиве **channelEndpoints** перечислены те из параметров, что отображаются внутри блоков «Channel N» (ограничены рамочкой на рис. 3.2 и 3.3), а в словаре **nonChannelEndpoints** перечислены те из них, что отображаются вне блоков (присутствуют только в расширенном режиме, см. рис. 3.3). Переменная состояния **AE** может быть

либо ссылкой на **AERegular** (значение по-умолчанию), либо ссылкой на **AEAdvanced** (в рассширенном режиме).

Для смены режима на расширенный код обработки нажатия на ссылку «Settings» инвертирует значение булевой переменной состояния **advancedDisabled** (нужна только для определения текущего режима), присваивает переменной состояния **AE** ссылку на **AEAdvanced**, вызывает обновление данных (описано в следующем элементе списка), меняет текст ссылки «Settings» на «Hide settings» и убирает класс **advanced-disabled** из тега с идентификатором (атрибутом **id**) **advanced-disabler**.

Для обратной смены режима также инвертируется **advancedDisabled**, **AE** присваивается обратно ссылка на **AERegular**, текст ссылки меняется обратно с «Hide settings» на «Settings», а тегу с идентификатором **advanced-disabler** опять присваивается класс **advanced-disabled**, приводящий к скрытию расширенных параметров.

Код, создающий константы **AERegular** и **AEAdvanced**, а также инициализирующий **AE**, представлен в листинге 34. Код смены режима с основного на расширенный и обратно представлен в листинге 50.

в) При загрузке Web-интерфейса, смене режима на расширенный или запросе на обновление всех отображаемых параметров путём нажатия кнопки «Update state», происходит обход всех значений массива **endpoints** переменной состояния **AE** с последующей отправкой API запросов для получения всех параметров по всем найденным путям. По мере получения ответов на отправленные запросы (все запросы отправляются параллельно) происходит обновление отображаемой информации. Для упрощения кода всегда обновляются все элементы, независимо от того, какие именно данные были получены.

Получением параметров Web-сервиса занимается функция **getSetState**. Обновлением отображаемой оператору информации занимается функция **updateShownState**. Получением списка API запросов, которые необходимо отправить для получения параметров Web-сервиса занимается функция **getStateDiff**, вызванная без аргументов. Все указанные функции представлены в листинге 46.

Отметим, что функция обновления **updateShownState** также сохраняет отображённые данные в переменной состояния **previousState**. Данная переменная служит для уменьшения числа запросов после изменения параметров Web-сервиса оператором.

г) При изменении какого-либо параметра Web-сервиса оператором для отправки соответствующего API запроса функция обработки события нажатия ввода оператором, либо нажатия кнопки «Set», либо изменения флажка для единственных двух параметров с флажком, обновляет переменную состояния **state** в соответствии с изменениями от оператора, после чего в POST API запросах отправляется разница между переменными состояния **previousState** и **state** (проверяется только разница в параметрах, перечисленных в переменной состояния **AE**). Этим занимаются те же функции **getSetState** и **getStateDiff**,

что и выше: первая отправляет запросы, вторая определяет, какие запросы нужно отправить.

Отметим, что функция, вызываемая при окончании обработки POST запроса Web-сервисом, затем вызывает GET запрос для получения нового значения параметра и обновления отображаемых данных: это сделано с целью проверки успешности установки заданного оператором значения.

Функция обработки события, вызываемая при описанных в начале пункта условиях, а также функция обновления **state**, представлены в листинге 51.

Функции отправки GET и POST API запросов **getAPIValue** и **setAPIValue** были вынесены в отдельный листинг 45.

```
var getInitialState = function(initialValue) {
    var channelState = { value1: initialValue, value2: initialValue };
    return {
        coef: {
            ki: deepCopy(channelState),
            kd: deepCopy(channelState),
            kp: deepCopy(channelState),
        },
        alarm: {
            a: deepCopy(channelState),
        },
        hyst: deepCopy(channelState),
        target: {
            value1: initialValue,
            value2: initialValue,
            enabled1: initialValue,
            enabled2: initialValue,
        },
        trid: {
            connect: {
                port: initialValue,
                baudrate: initialValue,
                parity: initialValue,
                stopbits: initialValue,
                bytesize: initialValue,
                mode: initialValue,
                slaveaddress: initialValue,
            },
            monitor: {
                interval: initialValue,
                data_points: initialValue,
            },
```

```
        },
    };
};

var initialState = getInitialState(null);

var state = deepCopy(initialState);
var previousState = deepCopy(state);
```

Листинг 32 — Код инициализации переменных состояния, используемых для работы с параметрами Web-сервиса

```
var deepCopy = function(obj) {
    return JSON.parse(JSON.stringify(obj));
};
```

Листинг 33 — Код функции **deepCopy**

```
var channelKeys = ['value1', 'value2'];
var allPorts = [];
var AEAdvanced = {
    endpoints: [
        [channelKeys, ['hyst']],
        [['value1', 'value2', 'enabled1', 'enabled2'], ['target']],
        [channelKeys, ['coef', 'ki']],
        [channelKeys, ['coef', 'kd']],
        [channelKeys, ['coef', 'kp']],
        [channelKeys, ['alarm', 'a']],
        [['port', 'baudrate', 'parity', 'stopbits', 'bytesize', 'mode',
          'slaveaddress'],
         ['trid', 'connect']],
        [['interval', 'data_points'], ['trid', 'monitor']],
    ],
};
AEAdvanced.channelEndpoints = [
    AEAdvanced.endpoints[0],
    AEAdvanced.endpoints[1],
    AEAdvanced.endpoints[2],
    AEAdvanced.endpoints[3],
    AEAdvanced.endpoints[4],
    AEAdvanced.endpoints[5],
];
AEAdvanced.nonChannelEndpoints = {
    connect: AEAdvanced.endpoints[AEAdvanced.endpoints.length - 2],
```

```
27      monitor: AEAdvanced.endpoints[AEAdvanced.endpoints.length - 1],
28  };
29  var AERegular = {
30      endpoints: [
31          [['value1', 'value2', 'enabled1', 'enabled2'], ['target']],
32      ],
33  };
34  AERegular.channelEndpoints = [
35      AERegular.endpoints[0],
36  ];
37  AERegular.nonChannelEndpoints = {};
38  var AE = AERegular;
```

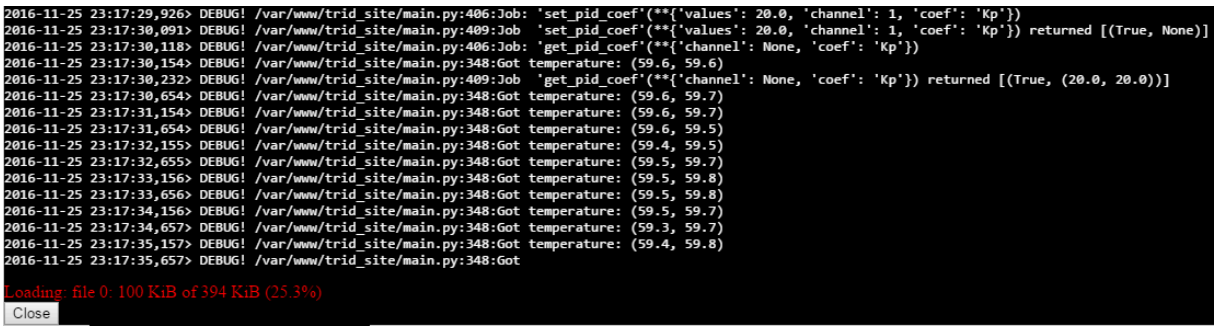
Листинг 34 — Код создания констант **AERegular** и **AEAdvanced** и инициализации переменной состояния **AE**

3.7.4.5 Функции режима показа журнала

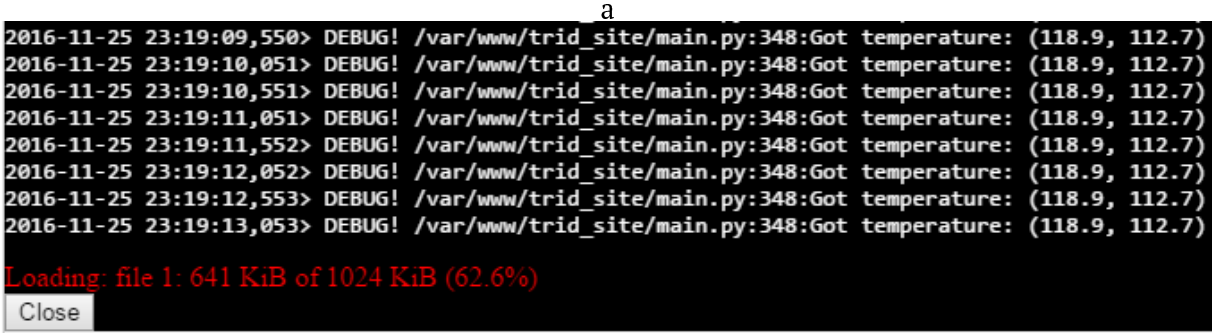
Для показа журнала применяются всего две функции: **getLog** получает журнал и отображает индикатор прогресса, **humanizeBytes** используется для показа числа загруженных байт загружаемого файла журнала и общего числа байт того же файла в индикаторе прогресса. Для того, чтобы ускорить показ журнала при использовании каналов со слабой пропускной способностью Web-сервис был настроен таким образом, что журнал сохраняется разбитым на 50 и менее файлов, каждый размером около 1 МиБ и менее. Функция показа журнала при этом загружает сначала последний файл и отображает его в конце экрана, потом предпоследний, который помещается до последнего, и так далее до загрузки всех файлов журнала. При этом всё время загрузки экран перематывается на конец журнала.

Смена режима при нажатии ссылки «Logs» происходит следующим образом: вызывается описанная выше функция «getLog» с двумя аргументами: DOM-элементом, тега-контейнера для журнала и DOM-элементом тега-контейнера индикатора прогресса, затем DOM-элементу тега с идентификатором (значением атрибута **id**) **logs** присваивается класс **logs**. Закрытие журнала по кнопке «Close» заключается всего лишь в присвоении тегу с идентификатором **logs** класса **logs-disabled**, который был у него изначально.

Внешний вид Web-интерфейса в режиме загрузки журнала показан на рис. 3.5. Код функций, используемых для отображения журнала приведён в листинге 47. Код функции-обработчика нажатия на ссылку «Logs» в основном и расширенном режимах приведён в листинге 52, код функции-обработчика нажатия на кнопку «Close», закрывающую журнал приведён в листинге 53



а



б

в

Рисунок 3.5 — Режим показа журнала:
а. в процессе загрузки последнего файла: последняя строка журнала ещё не загрузилась полностью, показан индикатор
б. в процессе загрузки предпоследнего файла
в. после загрузки последнего файла

3.7.4.6 Функции режима показа ошибки

Для показа ошибки используется всего одна функция **showError**, задача которой состоит в следующем:

а) Сохранить ошибку в JavaScript консоли вызовом метода браузера **console.log**.

б) Сохранить ошибку в контейнере с идентификатором (атрибутом **id**) **error-message** для показа пользователю.

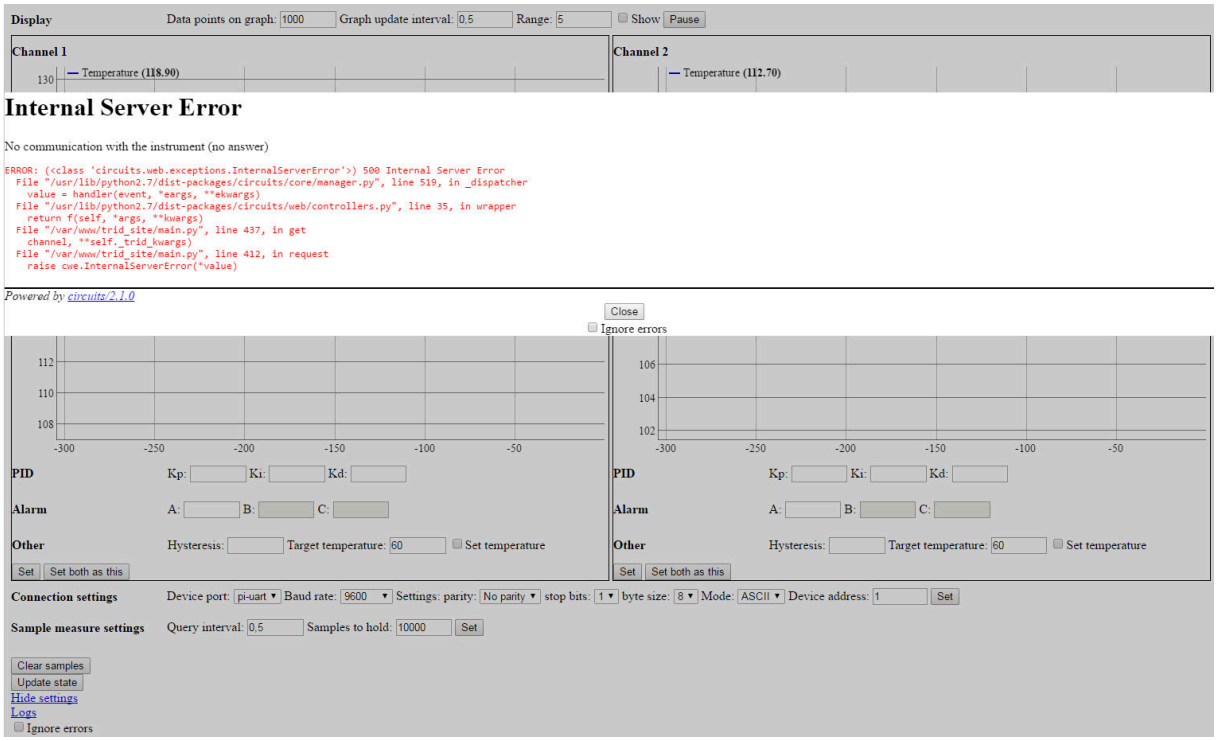


Рисунок 3.6 — Режим показа ошибки: ошибка, отображаемая при разрыве связи с ТРИД

в) Если переменная состояния **ignoreErrors** ложна (т.е. настройка «Ignore errors» не установлена, см. пункт 3.7.4.2), то включить режим отображения ошибки путём присвоения класса **error** контейнеру с идентификатором **error**.

Код данной функции представлен в листинге 35.

```
var eDiv = document.getElementById('error');

var showError = function(html, log) {
    console.log.apply(this, log);
    var eMsgDiv = document.getElementById('error-message');
    eMsgDiv.innerHTML = html;
    if (!ignoreErrors) {
        eDiv.classList = ['error'];
    }
};
```

Листинг 35 — Код функции **showError**

Закрытие режима отображения ошибки осуществляется просто восстановлением класса **error-disabled** у контейнера с идентификатором **error**. Код соответствующей фунции представлен в листинге 54.

Пример отображаемой ошибки показан на рис. 3.6.

3.7.4.7 Обработчики событий

Обработчики событий — функции, вызываемые из атрибутов **on***, примеры которых можно во множестве найти в пункте 3.7.1. Все имеющиеся в JavaScript коде обработчики событий являются небольшими функциями, обычно присваивающими классы различным DOM-элементам и/или вызывающие другие функции, иногда довольно объёмные. Большинство обработчиков было описано ранее:

а) Функции обработки изменения оператором сохраняемых настроек из листинга 48 описана в пункте 3.7.4.1.

б) Фукция обработки изменения оператором настройки «Ignore errors» из листинга 49 описана в пункте 3.7.4.2.

в) Функция изменения режима интерфейса с обычного на расширенный и обратно из листинга 50, а также функция обработки изменения параметров Web-сервиса оператором из листинга 51 описаны в пункте 3.7.4.4.

г) Функции показа и сокрытия журнала из листингов 52 и 53 соответственно описаны в пункте 3.7.4.5.

д) Функция закрытия режима отображения ошибки из листинга 54 описана в пункте 3.7.4.6.

Не описанными остались функции обработки нажатий на кнопки «Update state» и «Clear samples» (см. рис. 3.2), являющиеся типичными короткими представителями функция-обработчиков событий. Данные функции представлены в листинге 36

```
window.trid.update_everything = function() {
    getSetState();
    return false;
};
window.trid.clear_samples = function() {
    setAPIValue(url('/api/temp/clear'), {}, function() {
        data.length = 0;
    });
    return false;
};
```

Листинг 36 — Функции, обрабатывающие нажатия на кнопки «Update state» и «Clear samples»

Как можно видеть, обе данные функции возвращают **false**, впрочем, как и абсолютно все остальные функции-обработчики событий. Также из листингов в пункте 3.7.1 видно, что обработчики событий в HTML коде все определены примерно следующим

образом: **onchange="return window.trid.changed_graph_data_points()"**. Такой код является следствием того, что возвращаемое в атрибуте **on{event}** значение — истина или ложь — определяет, будут ли вызываться другие обработчики событий — в данном случае, стандартные обработчики. При возврате значения «ложь» (**false**) далее события не обрабатываются: всю обработку событий берёт на себя приведённый JavaScript код.

ПИД-регулятор — это устройство в управляющем контуре с обратной связью, используемое в системе автоматического управления для формирования управляющего сигнала[25]. Назначение регулятора в данном случае — поддержание заданного значения температуры ИС путём её подогрева с заданной регулятором интенсивностью, интенсивность нагрева задётся методом ШИМ.

Управляющий сигнал (скважность импульсов) при использовании ПИД-регулирования задётся как

$$S = \min\left\{\max\left\{\left(K'_p \Delta T(t) + K'_i \int_{t_0}^{t} \Delta T(\tau) d\tau + K'_d \frac{d\Delta T}{dt}\right); 0\right\}; 1\right\} \tag{4.1}$$

где S — скважность, может находится только в отрезке $[1, 0]$; K'_p, K'_i и K'_d — числа, соответствующие введённым оператором коэффициентам ПИД-регулирования; $\Delta T(t)$ — функция разницы между целевым значением температуры, введённым оператором, и измеряемым значением; а t_0 — некоторый момент во времени, используемый ПИД-регулятором для начала отсчёта. К сожалению, инструкция ТРИД не содержит подробной формулы, позволяющей определить способ интегрирования, момент t_0 или связь между введёнными оператором значениями ПИД-коэффициентов и числами K'_p, K'_i и K'_d, поэтому настройку ТРИД приходится осуществлять экспериментальным путём.

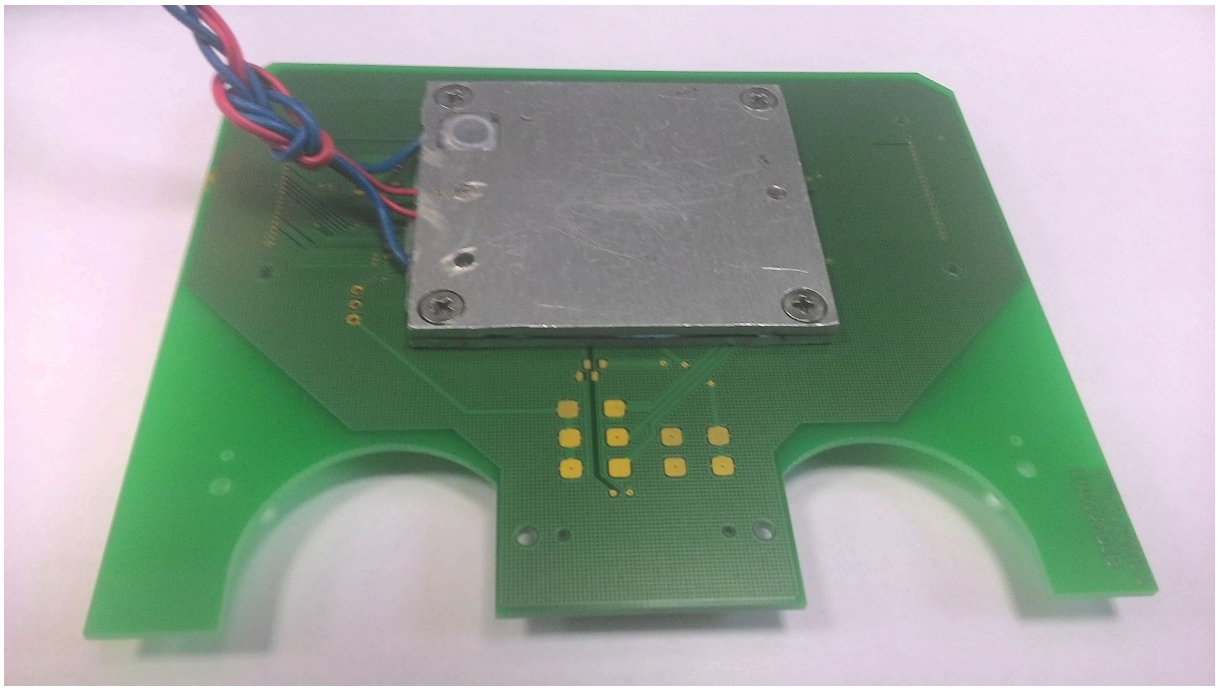


Рисунок 4.1 — Нераспаянная печатная плата с присоединённым нагревательным элементом

Для проверки работоспособности Web-интерфейса был собран тестовый стенд на котором производился нагрев нераспаянных печатных плат на обоих каналах (см. рис. 4.1).

Проверка работоспособности Web-интерфейса заключалась в проверке реакции системы на использование различных ПИД-коэффициентов: скорости нагрева (выхода на рабочий режим), величины перерегулирования при выходе на рабочий режим, величины перерегулирования при поддержании температуры. Для ускорения эксперимента перед выходом на рабочий режим температура стенда опускалась до $20 \div 35$ □ при помощи кубиков льда, после охлаждения стенда вода, полученная при расплавлении льда удалялась.

Отметим, что в Web-интерфейсе помимо ПИД-коэффициентов присутствуют параметры «Alarm A» (значение для аварийной сигнализации A) и «Hysteresis» (гистерезис). Данные параметры, хотя и могут быть установлены, не имеют никакого эффекта при используемых настройках ТРИД, однако информация об их работоспособности не может быть получена программным путём, её можно получить только при использовании оператором кнопочного интерфейса.

В ходе проверке работоспособности было проведено четыре эксперимента с тремя наборами ПИД-коэффициентов. Первые два заключались в проверке поддержания заданной температуры, вторые два проверяли выход на режим. Результаты экспериментов приведены в таблице 4.1, соответствующие графики приведены на рис. 4.2. Как видно из графиков, система реагирует на изменение ПИД-коэффициентов, а созданный интерфейс был проверен и одобрен оператором.

Таблица 4.1 — Результаты экспериментов

Рис.	K_p	K_i	K_d	Макс. размах при поддержании темп., □	Величина перерег. при выходе на режим, □	Скорость набора темп. при выходе на режим, $\frac{□}{c}$
а	50	5	1	1,8	N/A	N/A
б	20	2	5	1,5	N/A	N/A
в	30	10	1	0,3	5,0	0,94
г, д	10	3	5	2,1	3,9	3,36

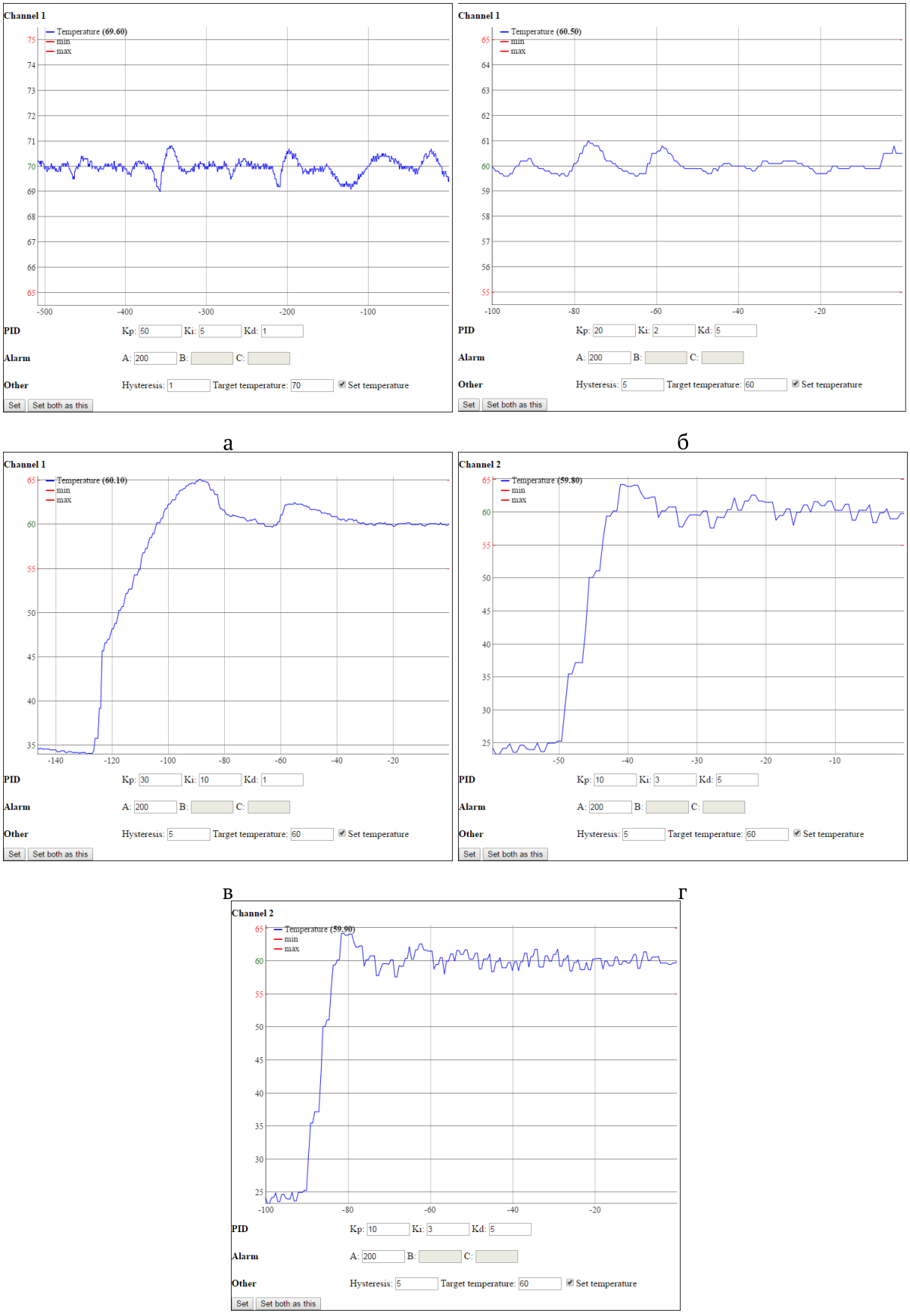


Рисунок 4.2 — Графики температуры в ходе экспериментов

ЗАКЛЮЧЕНИЕ

Был проанализирован процесс испытания СБИС на ускорителях лаборатории имени Г.Н. Флерова. Согласно результатом анализа были определены требования к модулю удалённого управления и способ удалённого доступа. На основе данных требований был определён тип управляющего модуля. Для выбранного типа модуля была выбрана операционная система и определены технологии, использованные для написания программного обеспечения. На основе выбранных технологий было разработано программное обеспечение, обеспечивающее удалённый доступ к ПИД-регулятору ТРИД РТП322.

Разработанная автоматизированная система удалённого контроля температуры СБИС была протестирована в лабораторных условиях. Готовится внедрение разработанной системы для проведения испытаний в реальных условиях АО «ЭНПО СПЭЛС».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ООО «Вектор-ПМ», Пермь. — ПИД-регулятор температуры двухканальный ТРИД РТП322, Руководство по эксплуатации ВПМ 421210.009-18 РЭ, 2012.

2. Multitasking library for the Atmel AVR processor. — Access mode: https://github.com/pietern/avr-tasks (online; accessed: 2017-02-20).

3. Arduino – ChatServer. — Access mode: https://www.arduino.cc/en/Tutorial/ChatServer (online; accessed: 2017-02-20).

4. Arduino/libraries/ESP8266AVRISP at master esp8266/Arduino. — Access mode: https://www.arduino.cc/en/Tutorial/WiFiChatServer (online; accessed: 2017-02-20).

5. Library files for building embedded network system using Atmel AVR Microcontroller and Microchip ENC28J60 Ethernet Controller. — Access mode: https://github.com/bprayudha/avr-enc28j60 (online; accessed: 2017-02-20).

6. Forum - Powered by Discuz! — Access mode: http://www.orangepi.org/orangepibbsen/forum.php (online; accessed: 2017-02-05).

7. banana pi single board computer official forum SinoVoip BPI team. — Access mode: http://forum.banana-pi.org/ (online; accessed: 2017-02-05).

8. Raspberry Pi Index page. — Access mode: https://www.raspberrypi.org/forums/ (online; accessed: 2017-02-05).

9. PINE64. — Access mode: https://forum.pine64.org/ (online; accessed: 2017-02-05).

10. Raspberry Pi Downloads. — Access mode: https://www.raspberrypi.org/downloads/ (online; accessed: 2017-02-10).

11. Raspberry Pi Downloads. — Access mode: https://www.raspberrypi.org/downloads/raspbian/ (online; accessed: 2017-02-10).

12. Historical trends in the usage of web servers for websites. — Access mode: https://w3techs.com/technologies/history_overview/web_server (online; accessed: 2017-02-10).

13. pyserial. — Access mode: https://pypi.python.org/pypi/pyserial (online; accessed: 2017-03-05).

14. MinimalModbus. — Access mode: https://pypi.python.org/pypi/MinimalModbus (online; accessed: 2017-03-05).

15. pylibmodbus. — Access mode: https://pypi.python.org/pypi/pylibmodbus (online; accessed: 2017-03-05).

16. modbus_tk. — Access mode: https://pypi.python.org/pypi/modbus_tk (online; accessed: 2017-03-05).

17. pymodbus. — Access mode: https://pypi.python.org/pypi/pymodbus (online; accessed: 2017-03-05).

18. uModbus. — Access mode: https://pypi.python.org/pypi/uModbus (online; accessed: 2017-03-05).

19. pymodbus github repository. — Access mode: https://github.com/bashwork/pymodbus (online; accessed: 2017-03-05).

20. circuits. — Access mode: https://pypi.python.org/pypi/circuits (online; accessed: 2017-03-05).

21. dygraphs.com. — Access mode: http://dygraphs.com/ (online; accessed: 2017-03-07).

22. Python : Any way to get one process to have a write lock and others to just read on parallel? — Access mode: http://stackoverflow.com/questions/16261902 (online; accessed: 2017-04-02).

23. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content : RFC : 7231 / RFC Editor ; Executor: Julian F. Reschke Roy T. Fielding : 2014. — Июнь. — P. 101. Access mode: https://tools.ietf.org/html/rfc7231.

24. The JavaScript Object Notation (JSON) Data Interchange Format : RFC : 7159 / RFC Editor ; Executor: Tim Bray : 2014. — Март. — P. 16. Access mode: https://tools.ietf.org/html/rfc7159.

25. ПИД-регулятор. — Access mode: https://ru.wikipedia.org/wiki/ПИД-регулятор (online; accessed: 2017-05-10).

Таблица А.1 — Константы со значениями по-умолчанию

Название	Значение	Описание
DEFAULT_MODE	asc	Режим протокола: текстовый.
DEFAULT_BAUD_RATE	9 600	Скорость обмена данными.
DEFAULT_SLAVE_ADDRESS	1	Адрес ТРИД.
DEFAULT_PORT_SETTINGS	**(PARITY_NONE, STOPITS_ONE, EIGHTBITS)**	Настройки порта: без битов чётности, с одним стоповым битом и 8-битным словом.
DEFAULT_TIMEOUT	0.5	Время ожидания ответа от ТРИД в секундах.

Таблица А.2 — Константы со поддерживаемыми значениями (словари или множества)

Название	Описание
SUPPORTED_BAUDRATES	Множество поддерживаемых скоростей обмена данными.
SUPPORTED_PORT_SETTINGS	Множество поддерживаемых настроек порта (кортежи с чётностью, числом стоповых бит и размером слова).
MODE_TRANSLATIONS	Словарь с поддерживаемыми режимами modbus (текстовым/бинарным).

Таблица А.3 — Методы класса TRID

Метод	Описание	Аргументы	Возвращаемое значение
I	II	III	IV
read_temperature	Получить показания температурных датчиков.	**channel** — канал: 1, 2 или **None** для получения с обоих каналов.	Одно значение (число), если канал 1 или 2, иначе кортеж с двумя значениями.

I	II	III	IV
get_target_temp	Получить установленную целевую температуру.	**channel** — канал: 1, 2 или **None** для получения с обоих каналов.	Одно значение (число), если канал 1 или 2, иначе кортеж с двумя значениями.
get_alarm	Получить установленное значение аварийной температуры.	**channel** — канал: 1, 2 или **None** для получения с обоих каналов; **alarm** — канал аварийной сигнализации: A, B или C.	Одно значение (число), если канал 1 или 2, иначе кортеж с двумя значениями.
get_hyst	Получить значение гистерезиса.	**channel** — канал: 1, 2 или **None** для получения с обоих каналов.	Одно значение (число), если канал 1 или 2, иначе кортеж с двумя значениями.
get_pid_coef	Получить значение ПИД коэффициента.	**channel** — канал: 1, 2 или **None** для получения с обоих каналов; **coef** — коэффициент: Kp, Ki или Kd.	Одно значение (число), если канал 1 или 2, иначе кортеж с двумя значениями.
set_target_temp	Установить значение целевой температуры.	**channel** — канал: 1, 2 или **None** для установки на обоих каналах; **values** — одно числовое значение, либо кортеж с двумя.	нет

I	II	III	IV
set_alarm	Установить аварийную сигнализацию.	**channel** — канал: 1, 2 или **None** для установки на обоих каналах; **values** — одно числовое значение, либо кортеж с двумя; **alarm** — канал аварийной сигнализации: A, B или C.	нет
set_hyst	Установить значение гистерезиса.	**channel** — канал: 1, 2 или **None** для установки на обоих каналах; **values** — одно числовое значение, либо кортеж с двумя.	нет
set_pid_coef	Установить значение коэффициента ПИД-регулирования.	**channel** — канал: 1, 2 или **None** для установки на обоих каналах; **values** — одно числовое значение, либо кортеж с двумя; **coef** — коэффициент: Kp, Ki или Kd.	нет

ПРИЛОЖЕНИЕ Б Реализация потока взаимодействия с ТРИД

Таблица Б.1 — Символы блок-схемы

Символ в блок-схеме	Тип символа	Тип значения	Описание
I	II	III	IV
e	Событие	-	Выход из цикла
t	Переменная	Рац. число	Время начала итерации
S	Блокировка	-	Блокировка настроек
интервал	Настройка	Неотриц. число	Интервал опроса ТРИД
i	Переменная	Неотриц. число	Сохранённая настройка: интервал опроса ТРИД
p	Переменная	Нат. число	Сохранённая настройка: число сохраняемых значений температуры
s	Переменная	Булево значение	Флаг: требуется ли установка целевой температуры
g	Переменная	Два рац. числа	Сохранённая настройка: целевая температура для каналов
L	Блокировка	-	Блокировка доступа к ТРИД
d	Переменная	Два рац. числа	Показания температурных датчиков ТРИД (два канала)
D	Блокировка	-	Блокировка доступа к данным на запись
темп.	Данные	(Время, (темп. кан. 1, темп. кан. 2))	Последние показания температурных датчиков
массив темп.	Данные	Список **temp.**	Все сохранённые показания температурных датчиков
w	Переменная	Рац. число	Время ожидания следующего задания в очереди
z	Переменная	Задание	Следующее задание: кортеж с событием, запросом к ТРИД, аргументами запроса (словарём) и контейнером для возврата результата

I	II	III	IV
r	Переменная	Пара: (флаг ошибки, результат запроса)	Результат запроса к ТРИД. Если запрос был успешен, то возвращается рац. число, пара рац. чисел либо **None**. В случае неудачи возвращается информация об ошибке.

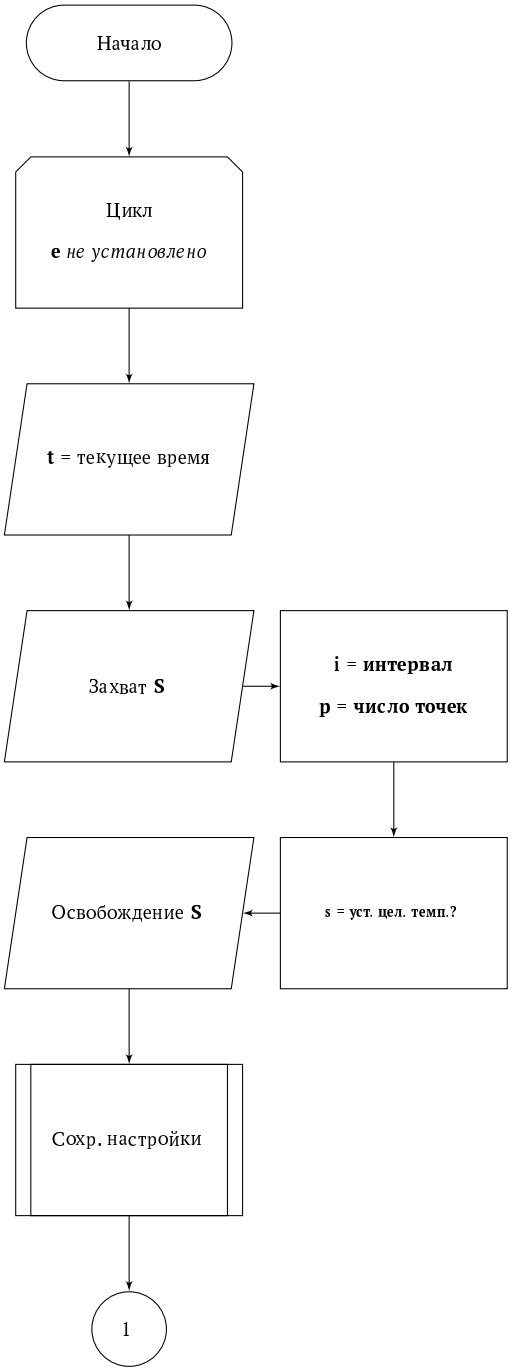


Рисунок Б.1 — Блок-схема потока взаимодействия с ТРИД, часть 1: получение основных настроек

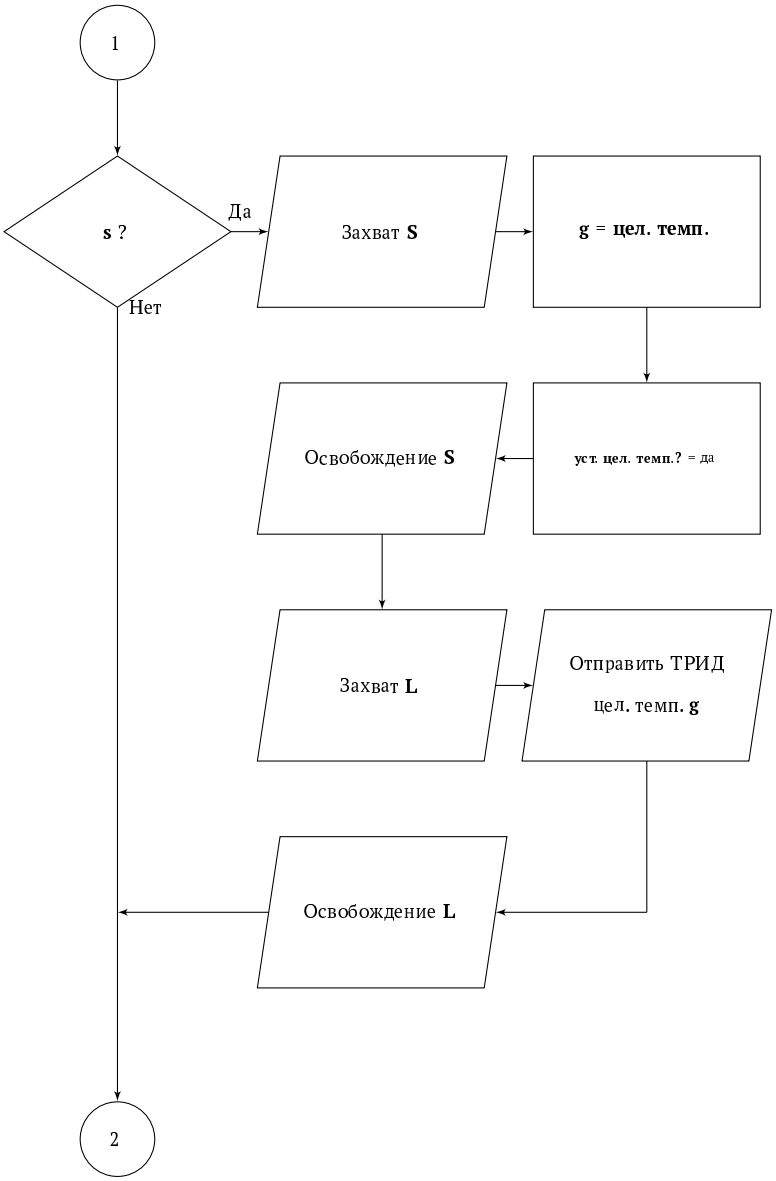


Рисунок Б.2 — Блок-схема потока взаимодействия с ТРИД, часть 2: установка целевой температуры

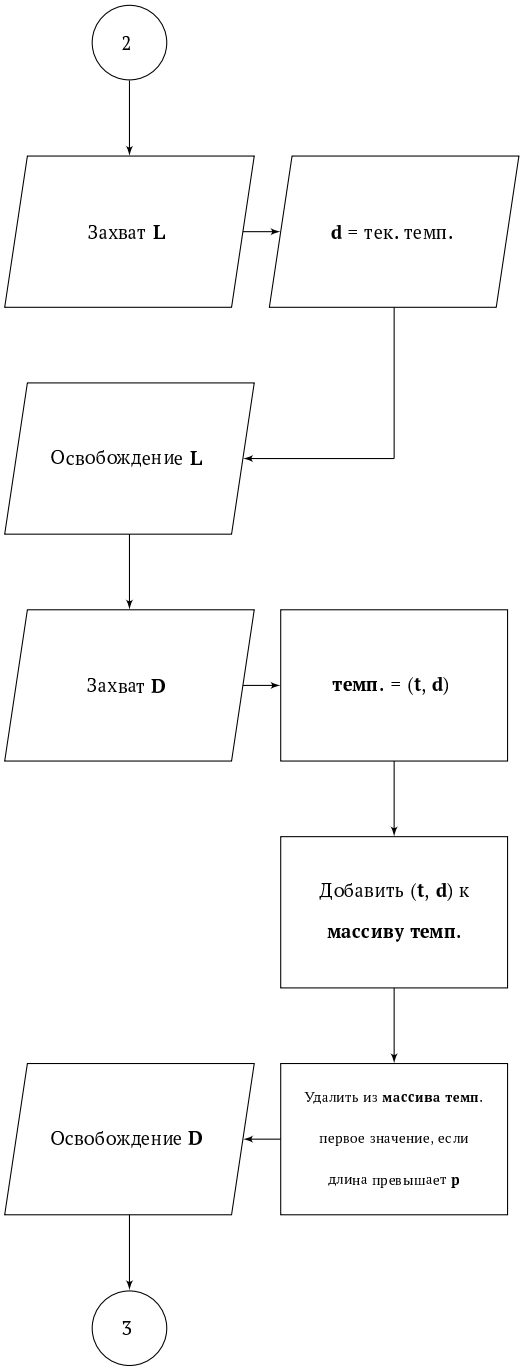


Рисунок Б.3 — Блок-схема потока взаимодействия с ТРИД, часть 3: считывание и сохранение показаний датчиков температуры

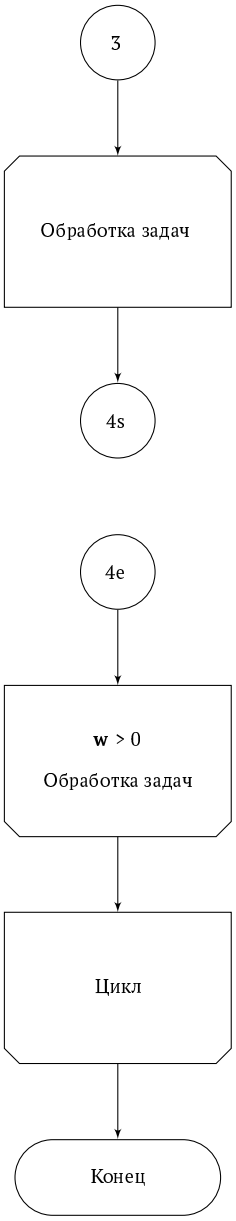


Рисунок Б.4 — Блок-схема потока взаимодействия с ТРИД, часть 4: цикл обработки задач (часть 1)

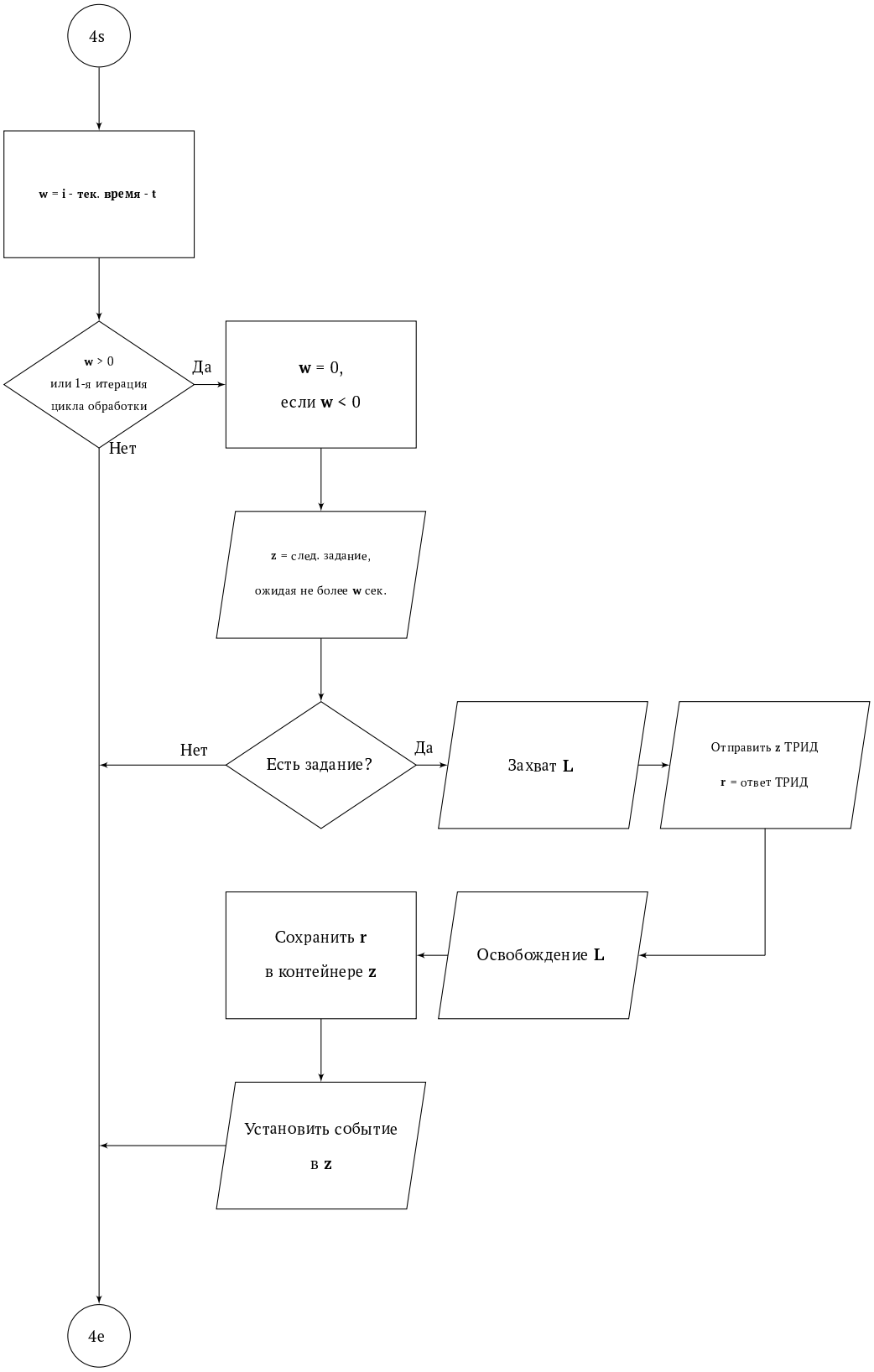


Рисунок Б.5 — Блок-схема потока взаимодействия с ТРИД, часть 5: цикл обработки задач: основная схема

```
class PIDControllerThread(threading.Thread):
    '''PID regulator controller thread class

    This is the only way to access PID regulator: this makes calls to
    regulator synchronous.'''
    daemon = True

    def __init__(self, state):
        super(PIDControllerThread, self).__init__()
        self.__state = state

    def run(self):
        min_wait = 0.0
        while not self.__state.shutdown_event.is_set():
            start_time = monotonic()
            with self.__state.state_lock:
                interval = self.__state.interval
                data_points = self.__state.data_points
                did_set_target_temp = self.__state.did_set_target_temp
            self.__state.save_settings()
            try:
                if not did_set_target_temp:
                    with self.__state.state_lock:
                        target_temp = self.__state.get_target_temp()
                        self.__state.did_set_target_temp = True
                    with self.__state.trid_lock:
                        self.__state.trid.set_target_temp(
                            values=target_temp)
                with self.__state.trid_lock:
                    temp = self.__state.trid.read_temperature()
            except Exception:
                logger.exception('Failed to get temperature')
            else:
                temperature = TimedTemperature(start_time, tuple(temp))
                with self.__state.data_lock.write_access:
                    self.__state.data.temperature = temperature
                    samples = self.__state.data.samples
                    if (len(samples) >= data_points):
                        del samples[0:len(samples) - data_points]
                    samples.append(temperature)
            first = True
            while True:
                wait_timeout = interval - (monotonic() - start_time)
                if wait_timeout <= min_wait:
```

```
                    if first:
                        wait_timeout = 0
                    else:
                        break
                first = False
                try:
                    finish_evt, meth, kwargs, ret_container = (
                        self.__state.job_queue.get(timeout=wait_timeout))
                except Queue.Empty:
                    break
                else:
                    try:
                        with self.__state.trid_lock:
                            ret = getattr(self.__state.trid,
                                          meth)(**kwargs)
                    except Exception:
                        logger.exception(
                            'Caught exception when processing job')
                        ret_container.append((False, (sys.exc_info()[1:])))
                    else:
                        ret_container.append((True, ret))
                    finish_evt.set()
            self.__state.shutdown_event.wait(
                max(interval - (monotonic() - start_time), min_wait))
```

Листинг 37 — Реализация потока взаимодействия

Таблица В.1 — API методы

Метод (без /api/)	Тип[1]	Описание	Параметры[2]	Возвращаемое значение[3]
I	II	III	IV	V
target/get	GET	Получить значение(я) целевой температуры.	channel	Словарь с ключами value1, value2, enabled1, enabled2
target	POST	Установить значение(я) целевой температуры и включить/выключить нагрев.	value1, value2, enabled1, enabled2	null
hyst/get	GET	Получить значение(я) ширины гистерезиса.	channel	Словарь с ключами value1, value2
hyst	POST	Установить значение(я) ширины гистерезиса.	value1, value2	null
alarm/a/get	GET	Получить значение аварийной температуры на канале A.	channel	Словарь с ключами value1, value2
alarm/a	POST	Установить значение аварийной температуры на канале A.	value1, value2	null
coef/kp/get	GET	Получить значение коэффициента K_p.	channel	Словарь с ключами value1, value2
coef/kp	POST	Установить значение коэффициента K_p.	value1, value2	null
coef/ki/get	GET	Получить значение коэффициента K_i.	channel	Словарь с ключами value1, value2

[1] GET, POST: тип HTTP запроса.

[2] См. табл. В.2

[3] См. табл. В.3 для возвращаемых «словарей», null означает, что при отсутствии ошибки возвращается строка null.

I	II	III	IV	V
coef/ki	POST	Установить значение коэффициента K_i.	value1, value2	null
coef/kd/get	GET	Получить значение коэффициента K_d.	channel	Словарь с ключами value1, value2
coef/kd	POST	Установить значение коэффициента K_d.	value1, value2	null
temp/get	GET	Получить показания датчиков температуры	channel	Кортеж с двумя значениями: **[timestamp, value1]**, **[timestamp, value2]** или **[timestamp, [value1, value2]]**.
temp/samples_since	GET	Получить список показаний датчиков температуры за данный период.	channel, timestamp	Список кортежей, идентичных кортежам, возвращаемым /api/temp/get.
temp/clear	POST	Сбросить сохранённый список показаний датчиков температуры.	нет	null
trid/connect/get	GET	Получить настройки соединения.	нет	Словарь с ключами mode, baudrate, slaveaddress, parity, stopbits, bytesize, port
trid/connect/list_ports	GET	Получить список известных портов UART, присутствующих в системе.	нет	Список строк.

I	II	III	IV	V
trid/connect	POST	Установить настройки соединения с ТРИД.	port, baudrate, slaveaddress, parity, stopbits, bytesize, mode	null
trid/monitor/get	GET	Получить настройки опроса ТРИД.	нет	Словарь с ключами interval, data_points
trid/monitor	POST	Установить настройки опроса ТРИД.	interval, data_points	null

Таблица В.2 — Описание параметров, используемых в API

Название параметра	Значения параметра	Описание параметра
I	II	III
channel	1, 2	Выбирает канал, значение на котором интересует вызывающего. В зависимости от этого в ответе будут присутствовать ключи value1 или value2, enabled1 или enabled2. Если данный параметр отсутствует, то в ответе будут присутствовать оба значения.
value1	Рациональное число	Значение для канала 1.
value2	Рациональное число	Значение для канала 2.
enabled1	true или false	Включение/отключение нагрева для канала 1.
enabled2	true или false	Включение/отключение нагрева для канала 2.
timestamp	Временная метка (рациональное число)	При запросе части собранных значений: минимальное время, после которого находятся интересные значения. При отсутствии параметра возвращаются все значения.

I	II	III
. port	Строка, одна из строк из списка /api/trid/connect/list_ports	UART порт, используемый для подключения к ТРИД.
baudrate	Натуральное число, одно из 9 600, 19 200, 28 800, 57 600, 115 200	Скорость соединения с ТРИД.
slaveaddress	Натуральное число от 1 до 255	Адрес ТРИД.
parity	N, E или O	Чётность.
bytesize	7 или 8	Размер одного слова, пересылаемого по UART.
stopbits	1 или 2	Количество бит, означающих конец слова.
mode	asc или rtu	Режим протокола modbus: текстовый (asc) или бинарный (rtu)
interval	Неотрицательное рациональное число	Интервал опроса ТРИД в секундах. Может быть нулём, в этом случае ТРИД опрашивается с максимальной доступной скоростью с учётам необходимости дождаться ответа до посылки следующего запроса.
data_points	Натуральное число	Максимальное количество хранимых значений температуры.

Таблица В.3 — Описание ключей словарей, используемых в возвращаемых значениях API

Название ключа	Значения	Описание
I	II	III
value1	Рациональное число	Значение для канала 1.
value2	Рациональное число	Значение для канала 2.
enabled1	true или false	Включение/отключение нагрева для канала 1.
enabled2	true или false	Включение/отключение нагрева для канала 2.

I	II	III
port	Строка, одна из строк из списка /api/trid/connect/list_ports	UART порт, используемый для подключения к ТРИД.
baudrate	Натуральное число, одно из 9 600, 19 200, 28 800, 57 600, 115 200	Скорость соединения с ТРИД.
slaveaddress	Натуральное число от 1 до 255	Адрес ТРИД.
parity	N, E или O	Чётность.
bytesize	7 или 8	Размер одного слова, пересылаемого по UART.
stopbits	1 или 2	Количество бит, означающих конец слова.
mode	asc или rtu	Режим протокола modbus: текстовый (asc) или бинарный (rtu)
interval	Неотрицательное рациональное число	Интервал опроса ТРИД в секундах. Может быть нулём, в этом случае ТРИД опрашивается с максимальной доступной скоростью с учётам необходимости дождаться ответа до посылки следующего запроса.
data_points	Натуральное число	Максимальное количество хранимых значений температуры.

Таблица В.4 — Экспериментально определённые граничные значения value1 и value2 ключей и параметров

API метод	Единицы измерения	Шаг	Минимум	Максимум
I	II	III	IV	V
/api/temp	°C	0,1	−270	2 500
/api/target	°C	0,1	−200	2 500
/api/hyst	°C	0,1	0,1	50
/api/alarm	°C	0,1	−200	2 500
/api/coef/kp	°C	0,1	0	3 000
/api/coef/ki	секунда	1	0	9 999
/api/coef/kd	секунда	0,1	0	999,9

```
class HysteresisGetSetAPI(TRIDGetSetAPI):
  channel = '/api/hyst'
  _trid_suffix = 'hyst'
  _trid_kwargs = {}
class AlarmAGetSetAPI(TRIDGetSetAPI):
  channel = '/api/alarm/a'
  _trid_suffix = 'alarm'
  _trid_kwargs = {'alarm': 'A'}
class PIDCoefKpGetSetAPI(TRIDGetSetAPI):
  channel = '/api/coef/kp'
  _trid_suffix = 'pid_coef'
  _trid_kwargs = {'coef': 'Kp'}
class PIDCoefKiGetSetAPI(TRIDGetSetAPI):
  channel = '/api/coef/ki'
  _trid_suffix = 'pid_coef'
  _trid_kwargs = {'coef': 'Ki'}
class PIDCoefKdGetSetAPI(TRIDGetSetAPI):
  channel = '/api/coef/kd'
  _trid_suffix = 'pid_coef'
  _trid_kwargs = {'coef': 'Kd'}
```

Листинг 38 — Код всех используемых классов-потомков **TRIDGetSetAPI**

```
import json
from circuits.web import Controller
import circuits.web.exceptions as cwe

class TRIDConnectAPI(Controller):
  channel = '/api/trid/connect'

  def __init__(self, thread_state, *args, **kwargs):
    super(TRIDConnectAPI, self).__init__(*args, **kwargs)
    self.__thread_state = thread_state

  def get(self):
    '''Get currently used settings'''
    with self.__thread_state.trid_lock:
      trid = self.__thread_state.trid
      port_settings = trid.port_settings
      settings = {
        'mode': trid.mode,
        'baudrate': trid.baudrate,
```

```
        'slaveaddress': trid.slaveaddress,
        'parity': port_settings.parity,
        'stopbits': port_settings.stopbits,
        'bytesize': port_settings.bytesize,
        'port': self.__thread_state.trid_port,
      }
    return json.dumps(settings)

  def list_ports(self):
    '''List available ports'''
    return json.dumps(list(PORTS.keys()))

  def POST(self,
      port='pi-uart',
      baudrate=trid.DEFAULT_BAUD_RATE,
      slaveaddress=trid.DEFAULT_SLAVE_ADDRESS,
      parity=trid.DEFAULT_PORT_SETTINGS.parity,
      stopbits=trid.DEFAULT_PORT_SETTINGS.stopbits,
      bytesize=trid.DEFAULT_PORT_SETTINGS.bytesize,
      mode=trid.DEFAULT_MODE):
    '''Connect to PID regulator using different settings

    :note: Uses default settings if not specified, not last used
      settings.'''
    if port not in PORTS:
      raise cwe.BadRequest(json.dumps({'error': 'Unknown port'}))
    try:
      baudrate = int(baudrate)
      slaveaddress = int(slaveaddress)
      stopbits = int(stopbits)
      bytesize = int(bytesize)
    except ValueError as e:
      raise cwe.BadRequest(json.dumps({
        'error': 'Argument not an integer'}))
    if baudrate not in trid.SUPPORTED_BAUDRATES:
      raise cwe.BadRequest(json.dumps({'error': 'Unsupported baud rate'}))
    if not (0x1 <= slaveaddress <= 0xFF):
      raise cwe.BadRequest(json.dumps({'error': 'Unsupported address'}))
    if bytesize not in {7, 8}:
      raise cwe.BadRequest(json.dumps({'error': 'Unsupported byte size'}))
    if parity not in set('NEO'):
      raise cwe.BadRequest(json.dumps({'error': 'Unsupported parity'}))
    port_settings = trid.PortSettings(bytesize=bytesize, parity=parity,
                    stopbits=stopbits)
    if port_settings not in trid.SUPPORTED_PORT_SETTINGS:
```

```
      raise cwe.BadRequest(json.dumps({
        'error': 'Unsupported port settings'}))
    if mode not in trid.MODE_TRANSLATIONS:
      raise cwe.BadRequest(json.dumps({'error': 'Unsupported mode'}))
    self.__thread_state.trid_connect(
      port,
      mode=mode, slaveaddress=slaveaddress, baudrate=baudrate,
      port_settings=port_settings)
    return 'null'
```

Листинг 41 — Код класса **TRIDConnectAPI**, обрабатывающего запросы к **/api/trid/connect**

```
import json
from circuits.web import Controller
import circuits.web.exceptions as cwe

class TRIDMonitorAPI(Controller):
  channel = '/api/trid/monitor'

  def __init__(self, thread_state, *args, **kwargs):
    super(TRIDMonitorAPI, self).__init__(*args, **kwargs)
    self.__thread_state = thread_state

  def POST(self, interval=None, data_points=None):
    '''Set monitor parameters: interval and amount of data points

    :param interval: Floating point value of new interval, in seconds.
    :param data_points: Integral value representing maximal amount of data
      points.'''
    if interval is not None:
      try:
        interval = float(interval)
      except ValueError:
        raise cwe.BadRequest(json.dumps({
          'error': 'Expected floating-point value'}))
    if data_points is not None:
      try:
        data_points = int(data_points)
      except ValueError:
        raise cwe.BadRequest(json.dumps({
          'error': 'Expected integral value'}))
    if (interval is not None and interval < 0) or (
        data_points is not None and data_points <= 0):
```

```
32        raise cwe.BadRequest(json.dumps({
33          'error': 'Expected positive value'}))
34      with self.__thread_state.data_lock.write_access:
35        if interval is not None:
36          self.__thread_state.interval = interval
37        if data_points is not None:
38          self.__thread_state.data_points = data_points
39        self.__thread_state.need_save_settings = True
40      return 'null'
41
42    def get(self):
43      '''Get currently used interval value and number of data points'''
44      with self.__thread_state.data_lock.read_access:
45        ret = {'interval': self.__thread_state.interval,
46           'data_points': self.__thread_state.data_points}
47      return json.dumps(ret)
```

Листинг 42 — Код класса **TRIDMonitorAPI**, обрабатывающего запросы к **/api/trid/monitor**

```
import bisect
import json
from circuits.web import Controller
import circuits.web.exceptions as cwe
PI_UART_PORT = '/dev/ttyAMA0'
'''Port used by UART on Raspberry Pi'''
PORTS = {'pi-uart': PI_UART_PORT}
'''Supported ports'''
class TemperatureAPI(Controller):
  channel = '/api/temp'
  def __init__(self, thread_state, *args, **kwargs):
    super(TemperatureAPI, self).__init__(*args, **kwargs)
    self.__thread_state = thread_state
    self += TemperatureClearAPI(thread_state)
  def get(self, channel=None):
    if channel not in {None, '1', '2'}:
      raise cwe.BadRequest(json.dumps({'error': 'Unexpected channel'}))
    with self.__thread_state.data_lock.read_access:
      ret = self.__thread_state.data.temperature
    if channel is not None:
      ret = (ret.timestamp, ret.temperature[int(channel) - 1])
  def samples_since(self, channel=None, timestamp=None):
    '''Get samples since the given time

    Returns all samples if time is not given.'''
    if channel not in {None, '1', '2'}:
      raise cwe.BadRequest(json.dumps({'error': 'Unexpected channel'}))
    if timestamp is not None:
      try:
        timestamp = float(timestamp)
      except ValueError:
        raise cwe.BadRequest(json.dumps({'error': 'Expected float'}))
    with self.__thread_state.data_lock.read_access:
      ret = self.__thread_state.data.samples[:]
    if timestamp is not None:
      temp = TimedTemperature(timestamp, (float('inf'), float('inf')))
      idx = bisect.bisect(ret, temp)
      del ret[:idx]
    if channel is not None:
      tidx = int(channel) - 1
      ret = [(i.timestamp, i.temperature[tidx]) for i in ret]
    return json.dumps(ret)
```

Листинг 39 — Код класса **TemperatureAPI**, обрабатывающего запросы к **/api/temp**

```
from circuits.web import Controller
class TemperatureClearAPI(Controller):
  '''/api/temp/clear API entry point

  :param PIDControllerThreadState thread_state:
    PID controller thread state.'''
  channel = '/api/temp/clear'

  def __init__(self, thread_state, *args, **kwargs):
    super(TemperatureClearAPI, self).__init__(*args, **kwargs)
    self.__thread_state = thread_state

  def POST(self):
    with self.__thread_state.data_lock.write_access:
      self.__thread_state.data.samples[:] = []
    return 'null'
```

Листинг 40 — Код класса **TemperatureClearAPI**, обрабатывающего запросы к **/api/temp/clear**

```
<div id="graph-form-div">
    <h1 id="graph-header" class="control-header">Display</h1>
    <form
      class="control"
      onsubmit="return window.trid.changed_graph_data_points()">
        Data points on graph: <input
         type="number"
         id="graph-max-data_points"
         step="any" min="0"
         class="num-input"
         onchange="return window.trid.changed_graph_data_points()"/>
    </form>
    <form
      class="control"
      onsubmit="return window.trid.changed_interval()">
        Graph update interval: <input
          type="number"
          id="graph-query-interval"
          step="any" min="0"
          class="num-input"
          onchange="return window.trid.changed_interval()"/>
    </form>
    <form class="control" onsubmit="return window.trid.changed_rng()">
        Range: <input
          type="number"
          id="graph-rng"
          class="num-input"
          step="0.1" min="0.1" max="6553.6"
          onchange="return window.trid.changed_rng()"/>
        <input
          id="graph-show-rng"
          type="checkbox"
          onchange="return window.trid.changed_rng()"
        /><label>Show</label>
    </form>
    <input
      type="button"
      value="Pause"
      id="graph-pause"
      onclick="return window.trid.graph_pause()"/>
</div>
```

Листинг 43 — HTML-код сохраняемых настроек Web-интерфейса

```
var maxDiffScale = 0.05;
var isNothing = function(obj) {
    return (obj === undefined || obj === null || obj === '');
};
var getGraphOptions = function(ch) {
    return {
        labels: ['Time', 'Temperature', 'min', 'max'],
        independentTicks: true,
        legend: 'always',
        valueFormatter: function(val,opts,seriesName,dygraph,row,col) {
            return val.toFixed(2);
        },
        legendFormatter: function(seriesData) {
            var showRange = getDisplayOption('showRange');
            if (seriesData.x == null) {
                // This happens when there's no selection and {legend:
                // 'always'} is set.
                var html = '';
                seriesData.series.forEach(function(series) {
                    if (!showRange && (series.label === 'min'
                                       || series.label === 'max')) {
                        return;
                    }
                    if (html !== '') {
                        html += '<br/>';
                    }
                    html += series.dashHTML + ' ' + series.labelHTML;
                    if (series.label === 'Temperature' && data.length>0) {
                        var lastTemp = data[data.length - 1][+ch];
                        html += ' <b>(' + lastTemp.toFixed(2) + ')</b>';
                    }
                });
                return html;
            }

            var html = this.getLabels()[0] + ': ' + seriesData.xHTML;
            seriesData.series.forEach(function(series) {
                if (!series.isVisible || series.label === 'min'
                    || series.label === 'max') {
                    return;
                }
                var labeledData = series.labelHTML + ': ' + series.yHTML;
                if (series.isHighlighted) {
```

```
                labeledData = '<b>' + labeledData + '</b>';
              }
              html += ' ' + series.dashHTML + ' ' + labeledData;
            });
            return html;
        },
        series: {
            min: {
                color: 'Red',
            },
            max: {
                color: 'Red',
            },
            Temperature: {
                color: 'Blue',
            },
        },
        axes: {
            y: {
                ticker: function(min, max, pixels, opts, dygraphs, vals) {
                    var ret = Dygraph.numericTicks(min, max, pixels, opts,
                                                   dygraphs, vals);
                    var key = 'value' + ch;
                    var maxDiff = maxDiffScale * (max - min);
                    if (!isNothing(state.target[key])) {
                        var p5idx = null;
                        var m5idx = null;
                        var zidx = null;
                        var z = state.target[key];
                        var rng = getDisplayOption('rng');
                        var showRange = getDisplayOption('showRange');
                        var newLabels = [
                            {
                                label: '<font color="#FF0000">' + (z + rng)
                                       + '</font>',
                                v: z + rng,
                            },
                            {
                                label: '<font color="#FF0000">' + (z - rng)
                                       + '</font>',
                                v: z - rng,
                            },
                            {
                                label: '<font color="#005500">' + z
                                       + '</font>',
```

```
                                v: z,
                            },
                        ];
                        var toRemove = [];
                        for (var i = 0; i < ret.length; i++) {
                            for (var j = 0; j < newLabels.length; j++) {
                                var diff=Math.abs(ret[i].v-newLabels[j].v);
                                if (diff < maxDiff) {
                                    toRemove.unshift(i);
                                    break;
                                }
                            }
                        }
                        for (var i = 0; i < toRemove.length; i++) {
                            ret.splice(toRemove[i], 1);
                        }
                        for (var i = 0; i < newLabels.length; i++) {
                            ret.push(newLabels[i]);
                        }
                    }
                    return ret;
                },
            },
        },
    };
};
var g1 = new Dygraph(document.getElementById('graph-channel-1'),
                     [[0, 0, 0, 0]],
                     getGraphOptions('1'));
var g2 = new Dygraph(document.getElementById('graph-channel-2'),
                     [[0, 0, 0, 0]],
                     getGraphOptions('2'));
```

Листинг 44 — Описание графика

```
var formatParams = function(pobj) {
    var params = '';
    for (var key in pobj) {
        if (pobj.hasOwnProperty(key) && key[0] !== '_') {
            if (params !== '') params += '&';
            var val = pobj[key];
            if (!isNothing(val)) {
                params += key + '=' + val;
            }
```

```
        }
    }
    return params;
};
var setAPIValue = function(request, pobj, callback) {
    var xhr = new XMLHttpRequest();
    xhr.open('POST', url(request), true);
    xhr.onreadystatechange = function() {
        if (xhr.readyState !== 4) return;
        if (callback) {
            callback(xhr.status, xhr.responseText);
        }
        if (xhr.status !== 200) {
            showError(xhr.responseText, [xhr]);
            return;
        }
    };
    xhr.setRequestHeader('Content-Type',
                         'application/x-www-form-urlencoded');
    xhr.send(formatParams(pobj));
};
var getAPIValue = function(setter, request) {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', url(request), true);
    xhr.onreadystatechange = function() {
        if (xhr.readyState !== 4) return;
        if (xhr.status !== 200) {
            showError(xhr.responseText, [xhr]);
            return;
        }
        setter(JSON.parse(xhr.responseText));
    };
    xhr.send();
};
```

Листинг 45 — Функции работы с API через XMLHttpRequest

```
var addCallProperty = function(calls, url, name, val) {
    if (calls[url] === undefined) {
        calls[url] = {};
    }
    calls[url][name] = val;
    if (calls.callsDict[url] === undefined) {
        calls.callsList.push(url);
```

```
    }
    calls.callsDict[url] = true;
};
var getStateDiff = function(previousState, state) {
    var calls = {callsList: [], callsDict: {}};
    for (var i = 0; i < AE.endpoints.length; i++) {
        var obj = state;
        var endpoint = AE.endpoints[i];
        var objKeys = endpoint[0];
        var objPath = endpoint[1];
        var APIPath = '/api/' + objPath.join('/');
        var obj = getObjAtPath(state, objPath);
        var previousObj = previousState;
        if (previousState !== null) {
            previousObj = getObjAtPath(previousState, objPath);
        }
        for (var j = 0; j < objKeys.length; j++) {
            var key = objKeys[j];
            if (isNothing(obj[key])
                || (previousState !== null
                    && obj[key] !== previousObj[key])) {
                addCallProperty(calls, APIPath, '_', {
                    path: objPath, keys: objKeys});
                addCallProperty(calls, APIPath, key, obj[key]);
            }
        }
    }
    return calls;
};
var updateShownState = function() {
    previousState = deepCopy(state);
    var select = document.getElementById(
        'trid-state-trid-connect-port-input');
    select.innerHTML = '';
    if (allPorts.length === 0) {
        ports = ['Default'];
    } else {
        ports = allPorts;
    }
    for (var i = 0 ; i < ports.length ; i++) {
        var opt = document.createElement('option');
        if (ports === allPorts) {
            opt.value = ports[i];
        } else {
            opt.value = '';
```

```
        }
        opt.innerText = ports[i];
        select.appendChild(opt);
    }

    for (var i = 0; i < AE.endpoints.length; i++) {
        var endpoint = AE.endpoints[i];
        var objKeys = endpoint[0];
        var objPath = endpoint[1];
        var idBase = 'trid-state-' + objPath.join('-');
        var obj = getObjAtPath(state, objPath);
        for (j = 0; j < objKeys.length; j++) {
            var key = objKeys[j];
            var id = idBase + '-' + key + '-input';
            var el = document.getElementById(id);
            setInputValue(el, obj[key]);
        }
    }
};
var getSetState = function(calls, doSet) {
    if (calls === undefined) {
        calls = getStateDiff(null, initialState);
    }
    var timeout = 0;
    for (var i = 0; i < calls.callsList.length; i++) {
        (function() {
            var APIBase = calls.callsList[i];
            var setPath = APIBase;
            var getPath = APIBase + '/get';
            var pobj = calls[APIBase];
            var update = function() {
                getAPIValue(function(response) {
                    var obj = getObjAtPath(state, pobj._.path);
                    var objKeys = pobj._.keys;
                    for (i = 0; i < objKeys.length; i++) {
                        obj[objKeys[i]] = response[objKeys[i]];
                    }
                    updateShownState();
                }, getPath);
            };
            if (doSet) {
                setTimeout(function() {
                    setAPIValue(setPath, pobj, update);
                }, timeout);
            } else {
```

```
                setTimeout(update, timeout);
            }
            timeout += 10;
        })();
    }
};
getAPIValue(function(response) {
    allPorts = response;
    updateShownState();
    getSetState();
}, '/api/trid/connect/list_ports');
```

Листинг 46 — Функции работы с параметрами Web-сервиса

```
var humanizeBytes = function(num, suffix, siPrefix) {
    var unitList = [[0, ''], [0, 'k'], [1, 'M'], [2, 'G'], [2, 'T'],
                    [2, 'P']];
    suffix = suffix || 'B';
    if (num === 0) {
        return 0 + ' ' + suffix;
    }
    var div = siPrefix ? 1000 : 1024;
    var exponent = Math.floor(Math.min(Math.log(num)/Math.log(div),
                                       unitList.length - 1));
    var quotient = num / (div ** exponent);
    var l = unitList[exponent];
    var decimals = l[0];
    var unit = l[1];
    if (unit !== '' && !siPrefix) {
        unit = unit.toUpperCase() + 'i';
    }
    return (quotient.toFixed(decimals) + ' ' + unit + suffix);
};

var getLog = function(logsContainer, logsProgress, num) {
    var u = '/logs/log';
    if (num === undefined) {
        logsContainer.innerHTML = '';
    } else {
        u += '.' + num;
    }
    var logsInnerContainer = document.createElement('pre');
    logsInnerContainer.id = 'logs-container-' + (num || 0);
    logsContainer.prepend(logsInnerContainer);
```

```
    var xhr = new XMLHttpRequest();
    xhr.open('GET', url(u));
    xhr.onprogress = function(event) {
        if (xhr.status === 404) return;
        logsProgress.innerText = (
                'Loading: file ' + (num || 0) + ': '
                + humanizeBytes(event.loaded)
                    + ' of ' + humanizeBytes(event.total)
                + ' ('
                    + (event.loaded / event.total * 100).toFixed(1)
                + '%)');
        logsInnerContainer.innerText = xhr.responseText;
        window.scrollTo(0, document.body.scrollHeight);
    };
    xhr.onreadystatechange = function() {
        if (xhr.readyState !== 4) return;
        logsProgress.innerText = '';
        if (xhr.status === 404) return;
        if (xhr.status !== 200) {
            showError(xhr.responseText, [xhr]);
            return;
        }
        logsInnerContainer.innerText = xhr.responseText;
        window.scrollTo(0, document.body.scrollHeight);
        if (num === undefined) {
            getLog(logsContainer, logsProgress, 1);
        } else {
            getLog(logsContainer, logsProgress, num + 1);
        }
    };
    xhr.send();
};
```

Листинг 47 — Функции отображения журнала

```
window.trid.changed_graph_data_points = function() {
    displayPreferInput.maxDataPoints = true;
    return false;
};
window.trid.changed_interval = function() {
    displayPreferInput.queryInterval = true;
    clearInterval(queryInterval);
    queryInterval = setInterval(
            queryTRID, getDisplayOption('queryInterval') * 1000);
```

```
    return false;
};
window.trid.changed_rng = function() {
    displayPreferInput.rng = true;
    displayPreferInput.showRange = true;
    updateGraph(g1, g2, data);
    return false;
};
```

Листинг 48 — Функция обработки изменения пользователем сохраняемых настроек

```
window.trid.toggle_ignore_errors = function(el) {
    ignoreErrors = el.checked;
    for (var i = 0; i < ignoreErrorsInputs.length; i++) {
        ignoreErrorsInputs[i].checked = ignoreErrors;
    }
    return false;
};
```

Листинг 49 — Функция обработки изменения флажков, соответствующих настройке «ignore errors»

```
var advancedDisabled = true;
var toggleAdvanced = function() {
    var link = document.getElementById('advanced-disabler-button');
    var disablerDiv = document.getElementById('advanced-disabler');
    if (advancedDisabled) {
        advancedDisabled = false;
        AE = AEAdvanced;
        getSetState();
        link.innerText = 'Hide settings';
        disablerDiv.classList = [];
    } else {
        advancedDisabled = true;
        AE = AERegular;
        link.innerText = 'Settings';
        disablerDiv.classList = ['advanced-disabled'];
    }
};
window.trid.toggle_advanced = function() {
    toggleAdvanced();
    return false;
};
```

Листинг 50 — Функция изменения режима: основной/расширенный

```
var collectState = function(el) {
    var id = el.id;
    var l = id.split('-');
    if (l[0] !== 'trid' || l[1] !== 'state') {
        return;
    }
    l.splice(0, 2);
    var etype = l[l.length - 1];
    var key = l[l.length - 2];
    l.length -= 2;
    if (etype === 'input') {
        var obj = getObjAtPath(state, l);
        obj[key] = getInputValue(el);
    } else if (etype === 'form' && l[l.length - 1] !== 'all') {
        // Must be a form for single values settings
        var obj = getObjAtPath(state, l);
        var inpId = 'trid-state-' + l.join('-') + '-' + key + '-input';
        var inpEl = document.getElementById(inpId);
        obj[key] = inpEl.value;
    } else if (etype === 'form' || etype === 'submit') {
        if (key !== 'all') {
            // Must be a trid-state-all-valueN-submit: set one channel
            // or trid-state-all-allN-submit: set both as this
            if (l.length !== 1 || l[0] !== 'all') {
                showError('Unexpected ID: ' + l.join('/'),
                          ['Unexpected ID', l]);
                return;
            }
            if (key === 'all1' || key === 'all2') {
                realKey = 'value' + key[3];
            } else {
                realKey = key;
            }
            for (var i = 0; i < AE.channelEndpoints.length; i++) {
                var endpoint = AE.channelEndpoints[i];
                var objPath = endpoint[1];
                var idBase = 'trid-state-' + objPath.join('-');
                var obj = getObjAtPath(state, objPath);
                var innerId = idBase + '-' + realKey + '-input';
                var innerEl = document.getElementById(innerId);
                var innerElEnabled = null;
                if (objPath.length === 1 && objPath[0] === 'target') {
                    innerElEnabled = document.getElementById(
```

```
                        idBase + '-' + realKey.replace('value', 'enabled')
                        + '-input');
                }
                if (realKey === key) {
                    obj[key] = innerEl.value;
                    if (innerElEnabled !== null) {
                        obj[key.replace('value', 'enabled')] = (
                            innerElEnabled.checked);
                    }
                } else {
                    obj.value1 = innerEl.value;
                    obj.value2 = innerEl.value;
                    if (innerElEnabled !== null) {
                        obj.enabled1 = innerElEnabled.checked;
                        obj.enabled2 = innerElEnabled.checked;
                    }
                }
            }
        } else {
            // Must be something like trid-state-trid-monitor-all-submit
            if (l.length !== 2 || l[0] !== 'trid'
                || AE.nonChannelEndpoints[l[1]] === undefined) {
                showError('Unexpected ID: ' + l.join('/'),
                          ['Unexpected ID', l]);
            }
            var endpoint = AE.nonChannelEndpoints[l[1]];
            var objKeys = endpoint[0];
            var objPath = endpoint[1];
            var idBase = 'trid-state-' + objPath.join('-');
            var obj = getObjAtPath(state, objPath);
            for (var i = 0; i < objKeys.length; i++) {
                var innerKey = objKeys[i];
                var innerId = idBase + '-' + innerKey + '-input';
                var innerEl = document.getElementById(innerId);
                obj[innerKey] = innerEl.value;
            }
        }
    }
};
window.trid.handle_event = function(el) {
    collectState(el);
    var calls = getStateDiff(previousState, state);
    updateShownState();
    getSetState(calls, true);
```

```
88      return false;
89  };
```

Листинг 51 — Функции, обрабатывающие изменение параметров Web-сервиса оператором

```
1   window.trid.show_logs = function() {
2       var logsDiv = document.getElementById('logs');
3       var logsContainer = document.getElementById('logs-container');
4       var logsProgress = document.getElementById('logs-progress');
5       getLog(logsContainer, logsProgress);
6       logsDiv.classList = ['logs'];
7       return false;
8   };
```

Листинг 52 — Функция, запускающая режим показа журнала

```
1   window.trid.hide_logs = function() {
2       var logsDiv = document.getElementById('logs');
3       logsDiv.classList = ['logs-disabled'];
4       return false;
5   };
```

Листинг 53 — Функция, обрабатывающая закрытие режима показа журнала

```
1   window.trid.hide_error = function() {
2       eDiv.classList = ['error-disabled'];
3       return false;
4   };
```

Листинг 54 — Функция, обрабатывающая закрытие режима отображения ошибки