Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский ядерный университет «МИФИ»

УДК 539.12.01

ОТЧЕТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

РАСШИРЕНИЕ СИСТЕМЫ ТЕМПЕРАТУРНОГО КОНТРОЛЯ СПЕКТРОМЕТРИЧЕСКОГО МАГНИТА СПАСЧАРМ

Научный руководитель
к.ф.-м.н.,
ведущий научный сотрудник НИЦ КИ-ИФВЭ _____
П. А. Семёнов

Выполнил работу _____
М. В. Гани

Москва, 2025

Содержание

ВВЕДЕНИЕ И ПОСТАНОВКА ЗАДАЧИ

Экспериментальная установка СПАСЧАРМ предназначена для исследований спиновых эффектов в адронных взаимодействиях на ускорительном комплексе У-70 (Протвино) и включает магнитный спектрометр с трековой системой. Согласно описанию установки, магнитный спектрометр содержит 57 плоскостей трековых детекторов и широкоапертурный спектрометрический магнит [1]. Работа установки в пучке требует устойчивого режима охлаждения магнитов и надёжной системы мониторинга параметров, влияющих на безопасность и воспроизводимость измерений.

Спектрометрический магнит М31 является водоохлаждаемым и имеет большую апертуру, обеспечивающую широкий телесный угол регистрации [2]. В диссертационной работе по системе управления СПАСЧАРМ приведены данные, что интеграл поля в центре магнита составляет порядка 1.5 Тл·м, а отвод тепла при работе осуществляется прокачкой обессоленной холодной воды под давлением порядка 11 атм [2]. Температурный мониторинг, как часть подсистемы медленного контроля, позволяет оперативно выявлять признаки ухудшения охлаждения (снижение протока, завоздушивание, загрязнение каналов), нештатные тепловые режимы и отказы датчиков.

Система медленного контроля СПАСЧАРМ построена на базе EPICS и реализует распределённую многоуровневую архитектуру [1, 2]. На уровне устройств работают специализированные микроконтроллерные модули; на уровне контроля — одноплатные компьютеры Raspberry Pi, на которых развёрнуты EPICS IOC; на уровне управления применяется операторская среда Control System Studio (CSS), а также сервисы архивирования и аварийной сигнализации [2]. Для обмена между уровнем устройств и уровнем контроля, по внутренней документации установки, используются промышленные протоколы Modbus RTU и CANbus [1].

Цель и задачи работы

Цель НИРС — разработка инженерно обоснованного проекта расширения подсистемы температурного контроля спектрометрического магнита М31 путём подключения второй гирлянды датчиков температуры, а также подготовка изменений в программно-аппаратной инфраструктуре (STM32 $\rightarrow$ Raspberry Pi/EPICS IOC) для интеграции новых каналов.

Задачи работы:

1. Выполнить обзор установки СПАСЧАРМ и места магнита М31 в составе магнитного спектрометра.
2. Проанализировать существующую архитектуру медленного контроля на базе EPICS, принципы PV-ориентированного интерфейса и требования к масштабируемости.
3. Проанализировать исходные коды микроконтроллерного уровня и IOC/драйверов EPICS и определить точки расширения для второй гирлянды датчиков.
4. Сформировать требования к набору PV, именованию, диагностике и аварийной сигнализации новых каналов.
5. Сформировать целевую концепцию: переход от потенциально блокирующих (синхронных) операций ввода-вывода к асинхронной модели, совместимой с EPICS.
6. Подготовить схемотехнические материалы и методику проверки работоспособности.

Состояние работ на момент промежуточного отчёта

Отчёт является промежуточным. На текущем этапе выполнено изучение доступной документации и программных артефактов (конфигурация IOC и пример прошивки STM32, схемы измерительного модуля). Сформулированы требования, предложена архитектура расширения и план испытаний. Реализация изменений в реально применяемом коде, изготовление и монтаж дополнительной гирлянды датчиков предполагаются на следующем этапе.

Принятые сокращения и термины

Сокращение	Расшифровка
EPICS	Experimental Physics and Industrial Control System — программная платформа для систем управления и медленного контроля
IOC	Input/Output Controller — приложение EPICS, обслуживающее набор PV и взаимодействующее с аппаратурой

PV	Process Variable — унифицированная переменная процесса, публикуемая IOC
CSS	Control System Studio — операторская среда и набор EPICS-клиентов
Modbus RTU	промышленный протокол обмена по последовательному интерфейсу (часто RS485)
CAN	Controller Area Network — промышленная шина обмена сообщениями
DS18B20/DS1820	цифровые датчики температуры семейства 1-Wire

ОБЗОР УСТАНОВКИ СПАСЧАРМ И МАГНИТНОГО СПЕКТРОМЕТРА

Состав установки и назначение спектрометрического магнита

СПАСЧАРМ включает подсистемы регистрации вторичных частиц и измерения их траекторий. Магнитный спектрометр служит для импульсного анализа заряженных частиц. На рис. 1 показана иллюстрация компоновки установки (по материалам внутреннего документа).

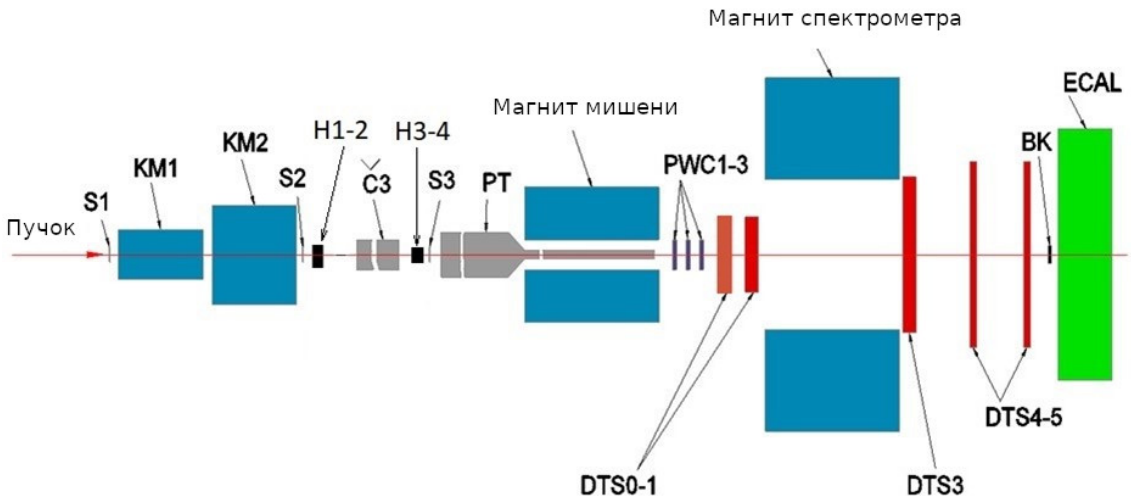

Рис. 1. Схема/компоновка установки СПАСЧАРМ (иллюстрация из внутреннего документа).

Параметры широкоапертурного магнита и его магнитного поля

Во внутреннем описании СПАСЧАРМ широкоапертурный магнит характеризуется следующими параметрами [1]:

- вертикальное поле формируется стальными полюсами с размерами порядка $X \times Z = 0.8 \times 1$ м^2;
- апертура магнита составляет порядка $X \times Y = 2.3 \times 1$ м^2 и с запасом перекрывает целевой угловой аксептанс не менее ± 110 мрад;
- при токе около 1 кА/виток поле в центре порядка 0.6 Тл.

Поле магнита существенно неоднородно; для использования в трекинге выполняются магнитные измерения и сопоставление расчётной модели с измеренной. В документации отмечено, что измерения компонент поля проводились в объёме порядка $X \times Y \times Z = 1.24 \times 0.84 \times 1.96$ м^3, а среднеквадратичное отклонение расчётов и измерений в рабочей апертуре составило около ± 1 мТл (примерно $\pm 0.2\%$ от поля в центре) [1]. Эти величины задают требования к воспроизводимости режимов магнита, в том числе по тепловой стабильности и устойчивости охлаждения.

Магнит M31: охлаждение и значимость температурного мониторинга

Фотография спектрометрического магнита M31 приведена на рис. 2. Магнит имеет водяное охлаждение; тепло отводится прокачкой обессоленной холодной воды под давлением порядка 11 атм [2]. В рамках эксплуатации требуется контроль температур как на уровне «вода/обмотки», так и в окрестности оборудования (влажность и температура воздуха), поскольку утечки и конденсация могут приводить к ухудшению изоляции и коррозии.

Рис. 2. Спектрометрический магнит M31 (по материалам диссертации).

Типовые причины отклонений температуры

С инженерной точки зрения температурные отклонения в водоохлаждаемом магните могут быть связаны со следующими факторами:

- снижение расхода воды, повышение температуры входной воды или засорение каналов;
- завоздушивание (падение теплопередачи) и локальные перегревы;
- неравномерность распределения расхода по параллельным каналам;
- деградация контакта датчика с поверхностью/трубопроводом, ошибочные показания из-за монтажа;
- отказ датчика или канала связи (обрыв, короткое замыкание, ошибки CRC, тайм-ауты).

Следовательно, проект расширения должен включать не только «добавление ещё датчиков», но и внятную схему диагностики и обработки статусов.

СИСТЕМА МЕДЛЕННОГО КОНТРОЛЯ СПАСЧАРМ НА БАЗЕ EPICS

Уровневая архитектура и роль PV

EPICS использует PV (Process Variable) как унифицированный интерфейс к измеряемым и управляемым параметрам. PV обслуживаются приложениями IOC (Input/Output Controller) и доступны операторским приложениям по сети (Channel Access). На рис. 3 приведён пример уровневой архитектуры системы управления/медленного контроля (по диссертации).

По внутренней документации СПАСЧАРМ, EPICS используется на двух уровнях — уровне управления и уровне контроля; на уровне устройств работают специализированные микроконтроллерные модули, а обмен между уровнем устройств и уровнем контроля реализован через Modbus RTU и CANbus [1]. Такой подход обеспечивает независимость операторского интерфейса от конкретной аппаратной реализации, однако повышает требования к качеству драйверов и устойчивости IOC при задержках или сбоях нижнего уровня.

Распределение IOC на Raspberry Pi

В диссертационной работе показано, что для установки СПАСЧАРМ характерна распределённая топология, где несколько Raspberry Pi обслуживают отдельные группы подсистем [2]. Пример подобной схемы приведён на

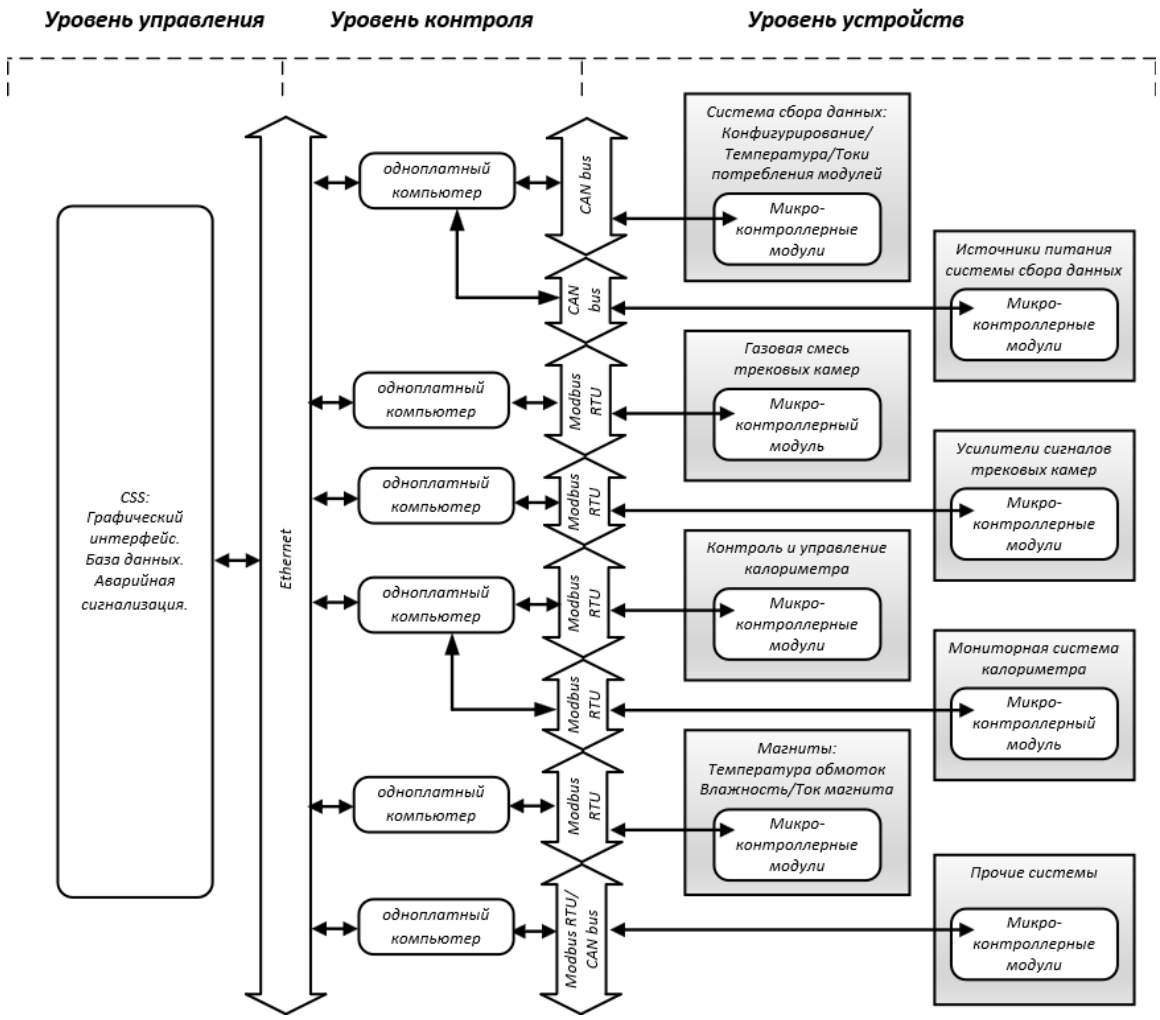

Рис. 3. Пример архитектуры системы управления/медленного контроля на базе EPICS (иллюстрация по диссертации).

рис. 4.

Для подключения промышленных шин (CAN, RS-485/Modbus) к Raspberry Pi применяются интерфейсные (мезонинные) платы с гальванической развязкой. На рис. 5 приведена блок-схема интерфейсного узла (DC/DC, контроллер CAN, изолированный драйвер Modbus), а на рис. 6 — пример схемы изолированного приёмопередатчика RS-485 (ISO3086T), применимого для Modbus RTU в условиях сильных помех и разности потенциалов земли.

С точки зрения расширения температурного контроля M31 это означает, что добавление каналов должно быть выполнено так, чтобы не нарушать существующие PV и не создавать «узких мест» на уровне конкретного IOC (CPU, блокировки ввода-вывода, частота опроса, время реакции).

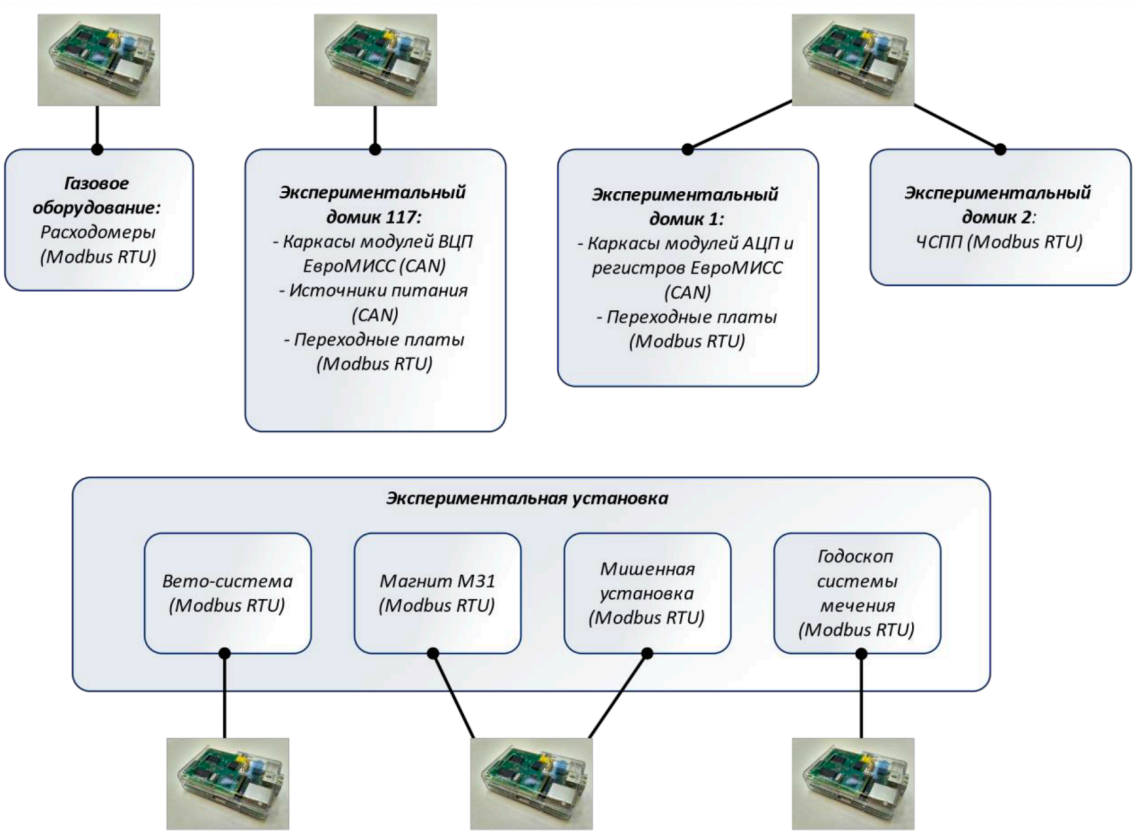

Рис. 4. Пример распределения контроллеров Raspberry Pi по подсистемам установки (иллюстрация по диссертации).

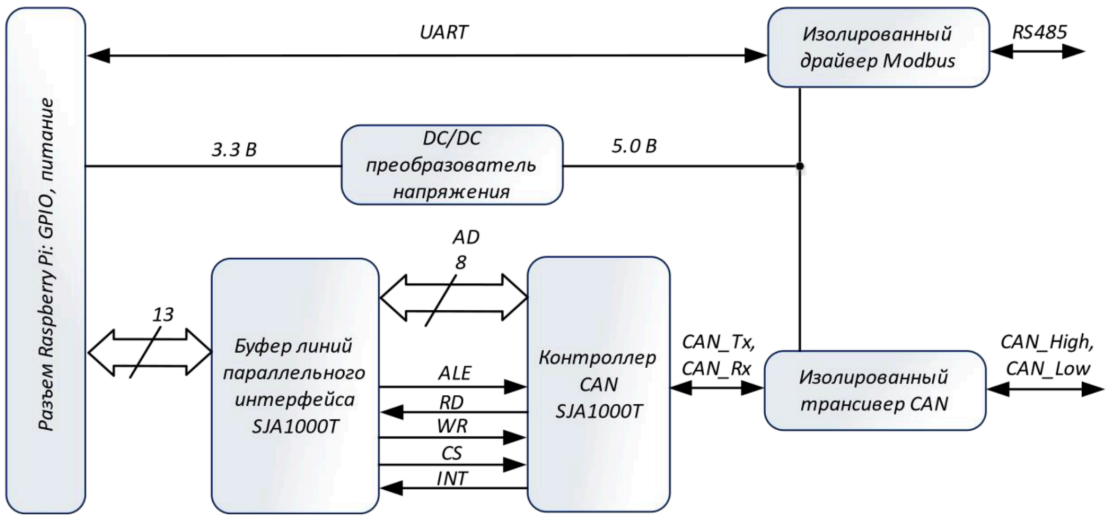

Рис. 5. Блок-схема интерфейсного узла Raspberry Pi для CAN и Modbus с гальванической развязкой (иллюстрация по диссертации).

Операторский уровень: CSS, архивирование и аварийная сигнализация

Операторская среда CSS обеспечивает визуализацию параметров, архивирование и аварийную сигнализацию. В диссертационной работе описано, что архиватор ArchiveEngine периодически сканирует PV и сохра-

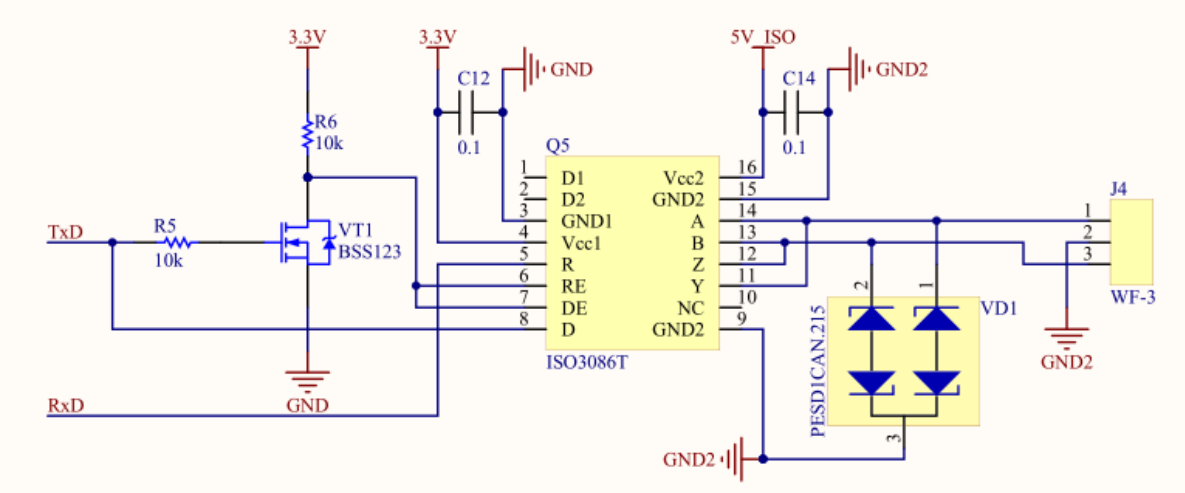

Рис. 6. Пример схемы изолированного RS-485 приёмопередатчика (ISO3086T) для Modbus RTU (иллюстрация по диссертации).

няет накопленные значения в базу данных (в качестве СУБД используется PostgreSQL) [2]. Пример панели CSS для контроля температурных каналов приведён на рис. 7.

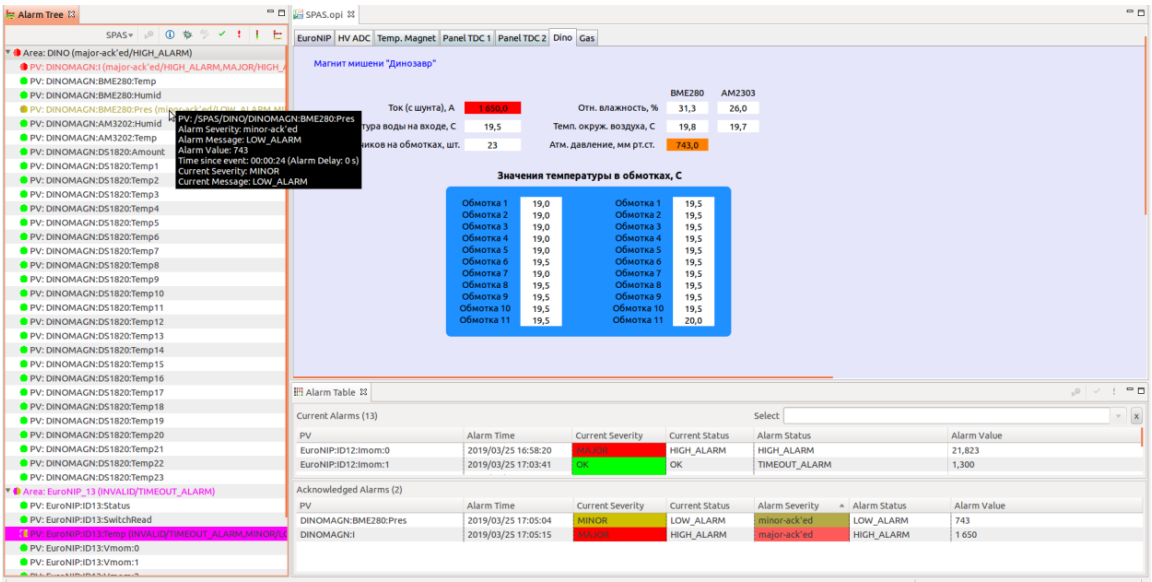

Рис. 7. Пример панели CSS для температурного мониторинга (иллюстрация по диссертации).

Для понимания потоков данных в системе управления существенна связка «IOC → CSS/архиватор → база данных». На рис. 8 показана функциональная схема взаимодействия операторских приложений CSS (архиватор, аварийная сигнализация, бортовой журнал) с базой данных PostgreSQL.

Пример интерфейса аварийной сигнализации CSS (Alarm Tree/Alarm Table) приведён на рис. 9. При расширении температурных каналов критично

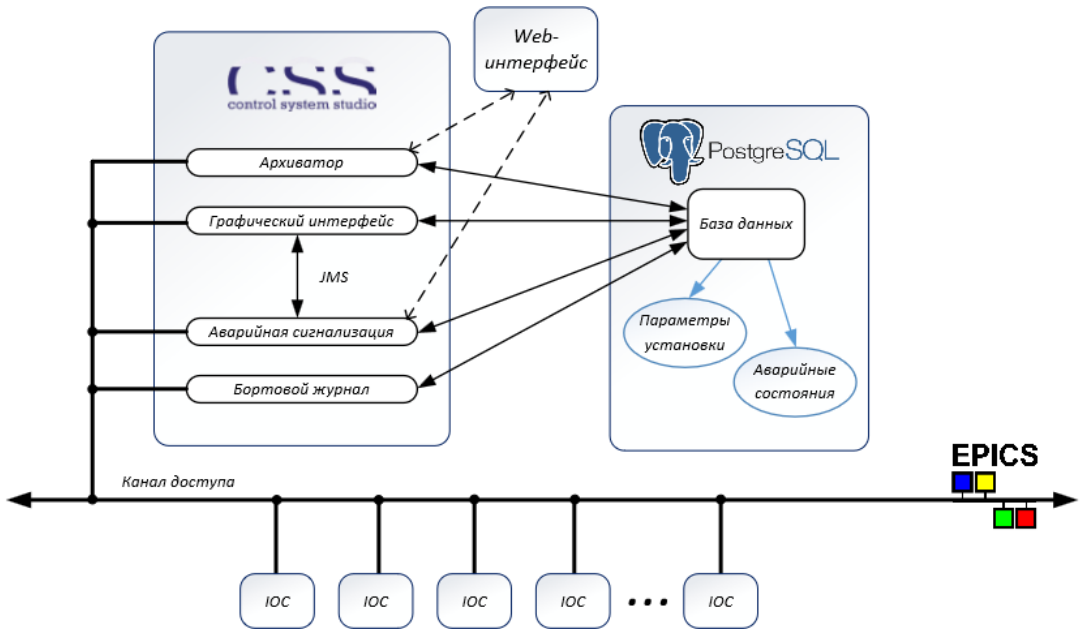

Рис. 8. Взаимодействие CSS, архиватора и базы данных PostgreSQL в системе управления СПАСЧАРМ (иллюстрация по диссертации).

обеспечить корректную выдачу статусов (например, INVALID/TIMEOUT) и уровней тревог, чтобы система сигнализации оставалась информативной и не деградировала в «шум».

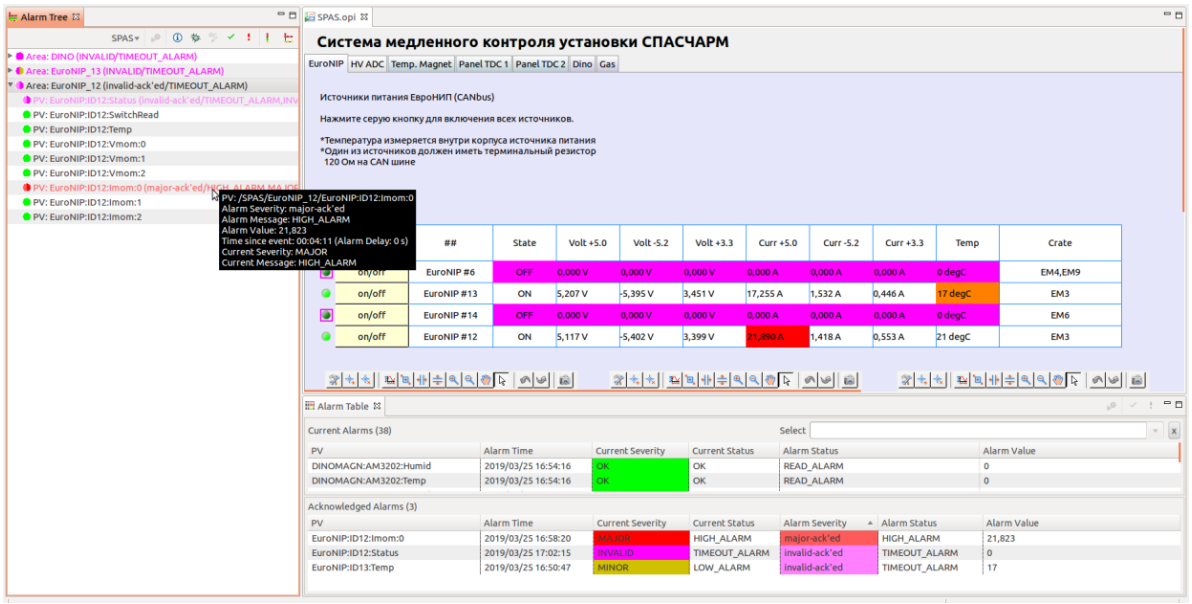

Рис. 9. Пример аварийной сигнализации в CSS (Alarm Tree/Alarm Table) в системе медленного контроля СПАСЧАРМ (иллюстрация по диссертации).

Для эксплуатационной пригодности расширения важно обеспечить корректные уровни тревог (MINOR/MAJOR), разумные пороги, а также единообразные статусы PV при ошибках связи. В противном случае добавление каналов приводит к «шуму» в аварийной сигнализации и снижению

полезности системы.

Почему в EPICS критична асинхронная модель ввода-вывода

В EPICS обработка записей (record processing) выполняется в рамках потоков IOC. Если функция чтения датчика блокирует (ожидание ответа по RS485, чтение файла, ожидание готовности преобразования температуры), это приводит к:

- разъезду периодов опроса (SCAN), пропускам обновлений и «рывкам» трендов;
- росту времени отклика операторского интерфейса;
- каскадным эффектам при увеличении числа каналов.

В диссертационной работе отмечается, что система управления СПА-СЧАРМ уже насчитывает порядка 700 PV и предполагается дальнейшее усложнение установки [2]. Поэтому при проектировании расширения температурного контроля M31 целесообразно закладывать асинхронную модель: взаимодействие с аппаратурой выполняется в выделенном потоке/очереди команд, а запись EPICS публикует последнее валидное значение и статус без длительных блокировок.

Цепочка формирования температурных PV для магнита M31

В текущей конфигурации на магните M31 уже установлена часть температурных датчиков (примерно половина контрольных точек на обмотках), объединённых в первую линию 1-Wire (гирлянда №1). Расширение системы предусматривает подключение оставшихся датчиков на **вторую независимую линию 1-Wire** (гирлянда №2), выведенную на **другой пин микроконтроллера STM32**. Raspberry Pi при этом не подключается к 1-Wire напрямую: опрос датчиков выполняется на STM32, а на уровень контроля данные передаются по **Modbus RTU** через **RS-485** (с гальванической развязкой по схеме измерительного модуля). На Raspberry Pi IOC EPICS считывает регистры Modbus через связку `asyn+modbus` и публикует значения в виде PV, доступных операторской среде CSS, архиватору и системе аварийной сигнализации.

Зачем нужны две независимые 1-Wire линии

Разделение датчиков на две независимые линии уменьшает суммарную длину/ёмкость каждой шины 1-Wire и тем самым повышает устойчивость об-

мена в условиях сильных электромагнитных помех возле силового магнита. Дополнительно это упрощает масштабирование: каждая линия может обслуживаться отдельным драйвером/каналом, а результаты могут агрегироваться в общей карте регистров Modbus.

Схема цепочки данных приведена на рис. 10. Такое разделение функций (опрос датчиков на STM32, публикация PV на IOC) позволяет:

- централизованно реализовать диагностику 1-Wire (CRC, контроль присутствия датчика, обработка ошибок);
- снизить чувствительность к электромагнитной обстановке около силового магнита за счёт промышленного интерфейса RS-485 с гальванической развязкой;
- изолировать «длинные» операции опроса датчиков от EPICS record processing.

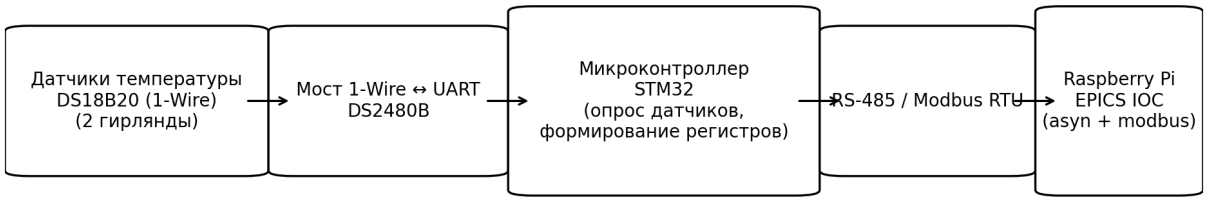

Рис. 10. Логическая цепочка формирования PV температурного мониторинга M31: две независимые шины 1-Wire (DS18B20) → драйвер/трансивер 1-Wire → STM32 → Modbus RTU/RS-485 → Raspberry Pi (IOC EPICS).

ПОДСИСТЕМА ТЕМПЕРАТУРНОГО КОНТРОЛЯ: КАНАЛЫ, ДАТЧИКИ, ДИАГНОСТИКА

Набор контролируемых величин

Минимально необходимый набор контролируемых величин для водоохлаждаемого магнита:

- температуры воды на входе/выходе и по отдельным ветвям охлаждения;
- параметры среды (температура и влажность) как диагностические индикаторы утечек/конденсации;
- косвенные параметры режима (например, ток магнита), позволяющие связать тепловую нагрузку с режимом [2].

Выбор датчиков и особенности 1-Wire

В предоставленных программных и схемотехнических материалах используются датчики семейства DS18B20/DS1820 (интерфейс 1-Wire). Преимущества: цифровой выход и наличие контроля целостности данных (CRC), возможность адресовать множество датчиков на одной линии. В исходном коде микроконтроллера (Приложение) преобразование выполняется с шагом $0.0625°C$ (12 бит), что соответствует стандартной разрядности DS18B20.

Монтаж и практические аспекты измерений

Фото примера установки температурных датчиков на трубопроводах/узлах охлаждения приведено на рис. 11.

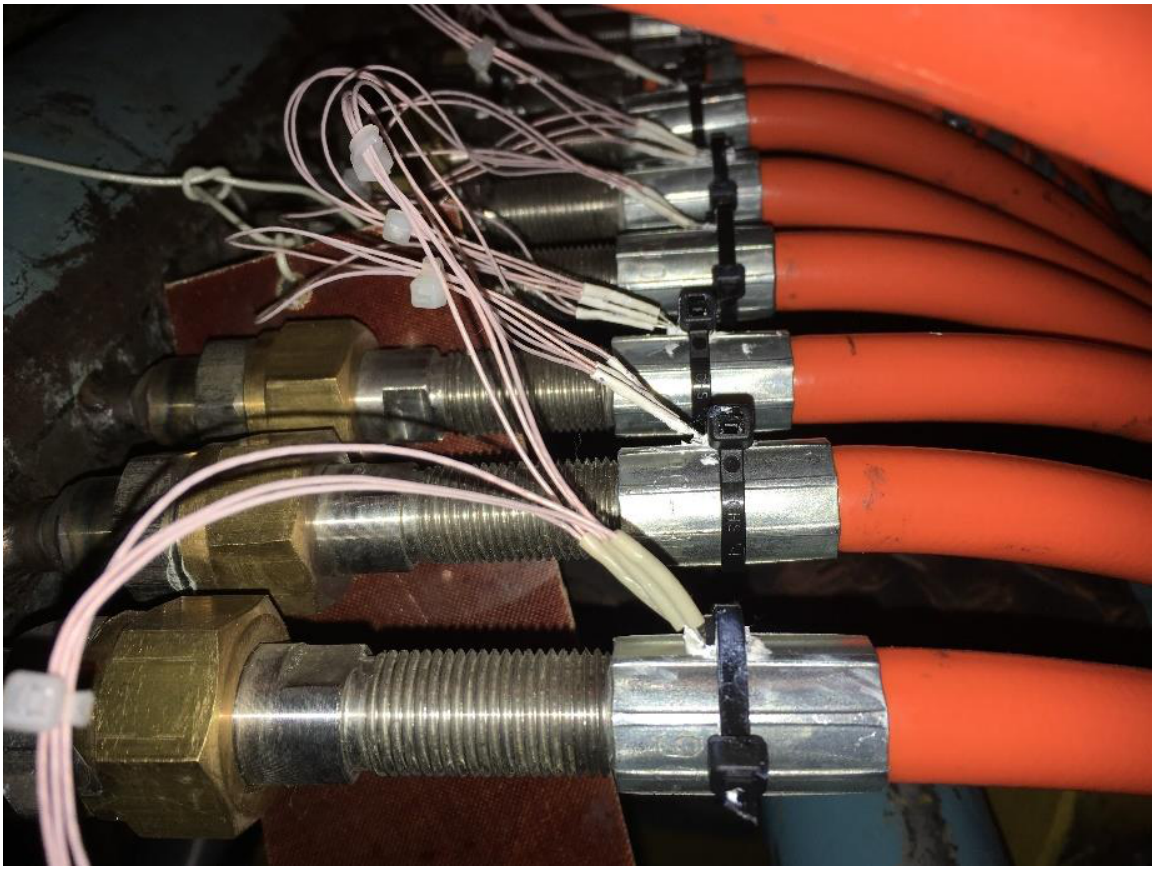

Рис. 11. Пример размещения температурных датчиков на трубопроводах охлаждения (иллюстрация по диссертации).

При проектировании второй гирлянды следует учитывать: надёжность механического контакта, термоизоляцию датчика от окружающего воздуха, стойкость кабеля к вибрациям/перемещениям, а также электромагнитную обстановку около мощного магнита и силовых кабелей.

АНАЛИЗ ПРОГРАММНО-АППАРАТНОЙ РЕАЛИЗАЦИИ И ПОДГОТОВКА РАСШИРЕНИЯ

Микроконтроллерный уровень: организация опроса 1-Wire и обмена по Modbus

Для нижнего уровня измерений в предоставленной прошивке STM32 реализовано:

- инициализация 1-Wire через мост DS2480B и поиск ROM-адресов датчиков;
- периодический цикл обновления температуры (таймер TIM2, период порядка нескольких секунд), чтобы избежать конфликтов с обменом по Modbus;
- обмен по Modbus RTU на скорости 9600 бод; выделение кадров по таймеру (TIM3) на интервале «тишины» порядка 3.5 символа.

Важное замечание из исходного кода: чтение датчиков температуры может занимать до сотен миллисекунд (характерно для конверсии DS18B20), что при синхронной реализации способно нарушать требования протокола Modbus и приводить к ошибкам. Поэтому в существующей реализации опрос датчиков вынесен в периодический обработчик, а код обработки Modbus отделён от «длинных» операций (Приложение).

Вывод для проекта второй гирлянды

При добавлении второй гирлянды возрастает число датчиков и, следовательно, длительность полного цикла опроса. Это требует либо оптимизации алгоритма (параллельная команда ConvertT для всех датчиков, затем пакетное чтение), либо увеличения периода обновления, либо аппаратного разделения на две независимые линии (например, две 1-Wire шины/два моста) с разнесением во времени, чтобы сохранить устойчивый Modbus-обмен.

Уточнение по исходным данным (магнит М29 как прототип)

Важно отметить, что файлы `dino.db`/`dino.substitutions` относятся к подсистеме температурного контроля магнита М29 (“Динозавр”). Поэтому в именах PV используется префикс `DINOMAGN`. В настоящей НИРС данный проект рассматривается как **прототип** и источник проверенных решений (структура `asyn+modbus`, организация тревог HIGH/HIHI, шаблоны

.substitutions). Для магнита M31 требуется переименовать PV и, при необходимости, скорректировать карту каналов; в настоящем отчёте в качестве рабочего префикса используется M31MAGN.

Пример структуры PV и порогов аварий

В базе dino.db (Приложение) температурные каналы DS1820 представлены как массив PV вида DINOMAGN:DS1820:Temp${CH} (IOC магнита M29). При переносе подхода на M31 целесообразно сохранить структуру массива, настройки порогов HIGH/HIHI (MINOR/MAJOR) и логику статусов, заменив префикс на рабочий M31MAGN:DS1820:Temp${CH} и уточнив привязку логических каналов к физическим датчикам двух линий 1-Wire (BUS1/BUS2).

Синхронные операции ввода-вывода и пути асинхронизации

Для температурного контроля M31 целевая схема обмена строится через Modbus RTU: Raspberry Pi взаимодействует с микроконтроллером STM32, который опрашивает датчики 1-Wire и предоставляет результаты измерений в виде регистров Modbus. Такой подход фактически переносит «длинные» и чувствительные к помехам операции (1-Wire) на специализированный нижний уровень и позволяет на стороне IOC использовать стандартный стек asyn+modbus.

Также присутствует пример device support EPICS для чтения датчиков 1-Wire через sysfs Linux (/sys/devices/w1_bus_master1//w1_slave) [3]. Данный вариант полезен как эталон алгоритма разбора формата w1_slave и проверки CRC, однако в условиях реальной установки (длины линий, электромагнитная обстановка, требования к промышленному интерфейсу) прямое подключение 1-Wire к Raspberry Pi не используется; обмен с датчиками организован через STM32 по Modbus RTU.

Оценка времени обмена Modbus и требований к периодам

Для скорости 9600 бод типичный ответ Modbus RTU на чтение 24 регистров (функция 3) имеет порядка 53 байт полезной посылки (адрес, функция, счётчик байтов, 48 байт данных, CRC), что соответствует ≈ 55 мс передачи (10 бит на байт) плюс запрос ≈ 8 мс. Даже без учёта пауз протокола и задержек драйвера один цикл «запрос+ответ» даёт десятки миллисекунд. При

большом числе PV и коротком SCAN это приводит к нагрузке на шину RS-485 и росту очереди asyn. Рационально синхронизировать период опроса Modbus с периодом обновления измерений на STM32 (в примере прошивки обновление параметров вынесено в таймер порядка нескольких секунд).

Предлагаемая асинхронная модель для расширения

С учётом фактической структуры IOC (asyn+Modbus) целесообразно опираться на стандартный механизм фонового опроса и кэширования данных:

1. На стороне **STM32** реализовать два независимых опроса 1-Wire (BUS1 и BUS2), но выдавать данные в виде **одной** (или двух) согласованной области регистров Modbus. Тем самым «длинные» операции ConvertT/чтение Scratchpad остаются вне критического пути обработки Modbus-кадров.
2. На стороне **IOC** настроить drvModbusAsynConfigure с **pollMsec**, который соответствует частоте обновления данных STM32 (например, 1000–3000 мс).
3. Для диагностики использовать отдельные PV (например, Amount на каждую линию) и/или битовую маску статусов в Modbus-регистрах (CRC OK, датчик найден, тайм-аут). На стороне EPICS эти признаки преобразуются в поля STAT/SEVR (INVALID, COMM) и в аварийные уровни CSS.
4. Для командных операций (сброс, принудительный поиск датчиков, смена периода опроса) использовать отдельные Modbus-регистры записи (функция 5/6/16), чтобы не вмешиваться в поток чтения.

Важный практический критерий: добавление каналов (вторая 1-Wire линия) не должно приводить к росту числа Modbus-транзакций *на запись*; вместо этого объём данных увеличивается внутри одного (или нескольких) пакетных чтений, обслуживаемых асинхронным драйвером.

СХЕМОТЕХНИЧЕСКАЯ ЧАСТЬ И ПРОЕКТ РАСШИРЕНИЯ ДЛЯ М31

Схема измерительного модуля и интерфейсы

На рис. 12 приведена схема измерительного модуля контроля температуры/параметров охлаждения. В составе присутствуют: микроконтроллер STM32, интерфейсы RS485/UART (Modbus RTU), CAN, модули влажности (AM2302, BME280) и температурный модуль на базе DS18B20/DS2480B.

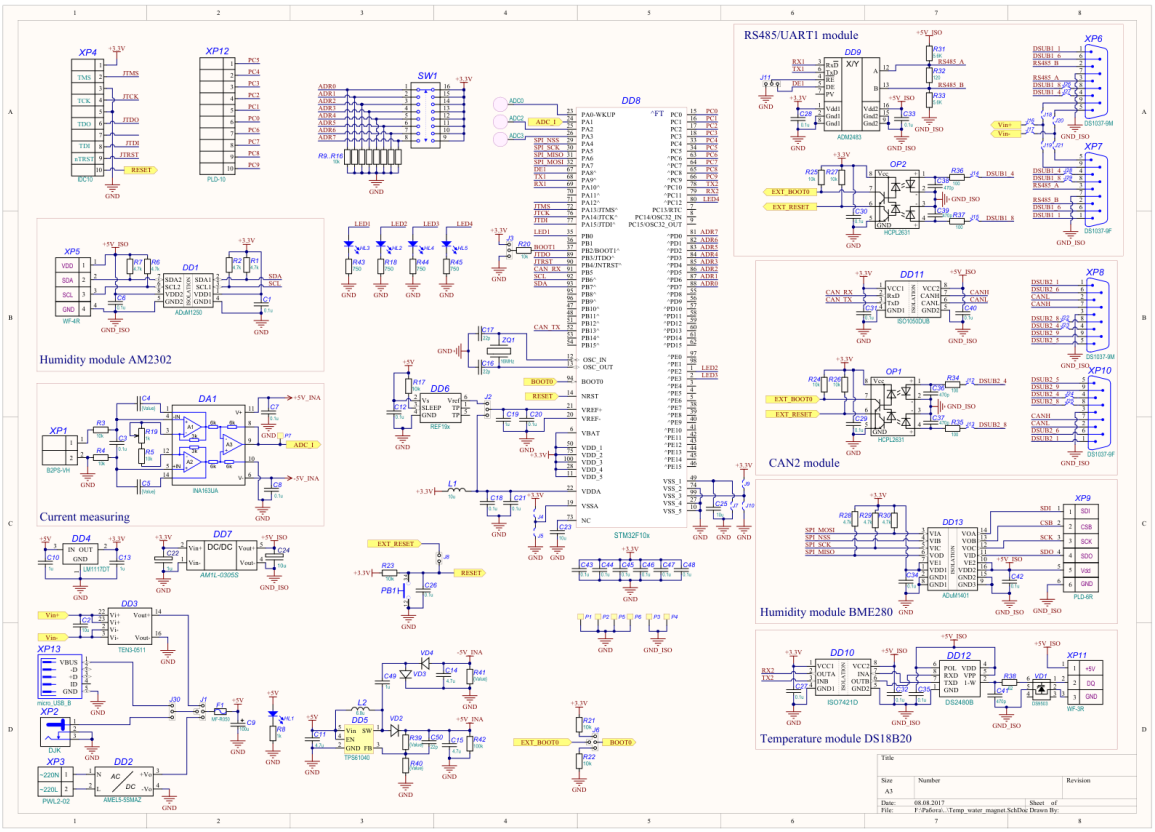

Рис. 12. Схема измерительного модуля контроля температуры и охлаждения.

Требования к второй гирлянде и интеграции в EPICS

Проект расширения второй гирляндой формулируется через требования к данным (PV), диагностике и совместимости:

- новые PV не должны ломать существующую структуру именования и панели CSS;
- для каждого датчика должны быть определены: физическое место установки, ROM-адрес (для 1-Wire), логический номер и соответствующий PV;

- должны быть определены пороги тревог и правила формирования статусов при ошибках (CRC/тайм-аут/обрыв);
- должен быть обеспечен режим деградации: при отказе части датчиков система продолжает предоставлять данные по исправным каналам.

Предложение по структуре PV для M31

С учётом примера `dino.db` предлагается шаблон именования для M31:

- `M31:WATER:IN_TEMP`, `M31:WATER:OUT_TEMP` — температуры воды на входе/выходе;
- `M31:COIL:TEMP1`–`M31:COIL:TEMP`N — температуры по точкам на обмотках/корпусе;
- `M31:ENV:TEMP`, `M31:ENV:HUMID` — параметры среды.

Точная карта каналов (количество датчиков, их размещение и ROM-адреса) уточняется по результатам монтажа и стендовых испытаний.

Карта регистров Modbus и таблица PV для второй гирлянды

Анализ конфигурации IOC показывает, что температурные значения в текущем проекте считываются из блока регистров Modbus (функция 3), заданного в `st.cmd` вызовом `drvModbusAsynConfigure()` с параметрами `startAddress=84` и `length=24` (Приложение). Первый регистр блока (смещение 0) содержит количество обнаруженных датчиков (`Amount`), а регистры со смещениями 1–23 соответствуют температурам (масштабирование в PV выполняется через коэффициент `ASLO`). В `dino.substitutions` используется перестановка логических каналов (например, канал 23 привязан к смещению 1), чтобы согласовать порядок выдачи измерений микроконтроллером с логикой нумерации на уровне CSS [3].

С учётом того, что на M31 уже установлена лишь часть датчиков, а оставшиеся подключаются как вторая независимая линия 1-Wire на другом пине STM32, возможны два инженерных варианта формирования карты регистров Modbus:

- **Вариант A:** STM32 агрегирует показания двух 1-Wire линий в **единый** массив температур и продолжает отдавать их в прежнем блоке (84, длина 24). Тогда конфигурация IOC/CSS может не меняться, а расширение сводится к изменению прошивки и уточнению физической карты датчиков.

- **Вариант B:** каждая 1-Wire линия отдаётся отдельным непрерывным блоком регистров (например, BUS1: 84..107, BUS2: 108..131). Такой вариант прозрачен для диагностики (можно иметь две переменные `Amount`), но требует добавить второй Modbus-порт и PV/шаблоны на стороне IOC.

Таблица соответствия каналов для линии BUS1: прототип M29 и переименование для M31

Табл. 2 составлена по файлам `dino.db`/`dino.substitutions` и отражает фактическое соответствие смещений внутри Modbus-блока для магнита M29 (“Динозавр”) **вместе** с предлагаемым переименованием PV для магнита M31. Иными словами, в проекте M31 сохраняется структура данных и offsets, но меняются префиксы и физическая интерпретация каналов (датчики на обмотках M31).

Таблица 2. Сопоставление PV прототипа (M29, “Динозавр”) и проекта (M31, BUS1) с соответствием смещений в Modbus-блоке.

#	PV (M29, прототип)	PV (M31, проект)	Offset	Основание
–	DINOMAGN:DS1820:Amount	M31MAGN:DS1820:Amount	0	dino.db
1	DINOMAGN:DS1820:Temp1	M31MAGN:DS1820:Temp1	2	dino.subst
2	DINOMAGN:DS1820:Temp2	M31MAGN:DS1820:Temp2	3	dino.subst
3	DINOMAGN:DS1820:Temp3	M31MAGN:DS1820:Temp3	4	dino.subst
4	DINOMAGN:DS1820:Temp4	M31MAGN:DS1820:Temp4	5	dino.subst
5	DINOMAGN:DS1820:Temp5	M31MAGN:DS1820:Temp5	6	dino.subst
6	DINOMAGN:DS1820:Temp6	M31MAGN:DS1820:Temp6	7	dino.subst
7	DINOMAGN:DS1820:Temp7	M31MAGN:DS1820:Temp7	8	dino.subst
8	DINOMAGN:DS1820:Temp8	M31MAGN:DS1820:Temp8	9	dino.subst
9	DINOMAGN:DS1820:Temp9	M31MAGN:DS1820:Temp9	10	dino.subst
10	DINOMAGN:DS1820:Temp10	M31MAGN:DS1820:Temp10	11	dino.subst
11	DINOMAGN:DS1820:Temp11	M31MAGN:DS1820:Temp11	12	dino.subst
12	DINOMAGN:DS1820:Temp12	M31MAGN:DS1820:Temp12	13	dino.subst
13	DINOMAGN:DS1820:Temp13		14	dino.subst
14	DINOMAGN:DS1820:Temp14		15	dino.subst
15	DINOMAGN:DS1820:Temp15		16	dino.subst
16	DINOMAGN:DS1820:Temp16		17	dino.subst
17	DINOMAGN:DS1820:Temp17		18	dino.subst
18	DINOMAGN:DS1820:Temp18		19	dino.subst

Продолжение на следующей странице

#	PV (M29, прототип)	PV (M31, проект)	Offset	Основание
19	DINOMAGN:DS1820:Temp19		20	dino.subst
20	DINOMAGN:DS1820:Temp20		21	dino.subst
21	DINOMAGN:DS1820:Temp21		22	dino.subst
22	DINOMAGN:DS1820:Temp22		23	dino.subst
23	DINOMAGN:DS1820:Temp23		1	dino.subst

Таблица PV для второй линии датчиков (BUS2, M31) в варианте B

Вторая гирлянда датчиков M31 соответствует **оставшимся** датчикам температуры, которые подключаются к отдельной линии 1-Wire (другой пин/канал STM32) и опрашиваются независимо от BUS1. Если выбрать вариант B с отдельным Modbus-блоком для BUS2, удобно использовать тот же формат `Amount + Temp[1..N]`. Табл. 3 приводит рекомендуемое соответствие (без перестановок; при необходимости перестановка может быть введена по аналогии с BUS1 для согласования с принятым порядком каналов на панелях CSS).

Таблица 3. Предлагаемая таблица PV для второй линии датчиков DS1820 (BUS2, M31) и соответствие смещениям в Modbus-блоке (проект).

#	PV (M31, проект)	Offset	Примечание
–	M31MAGN:DS1820_BUS2:Amount	0	Количество обнаруженных датчиков линии BUS2 (оставшиеся датчики M31)
13	M31MAGN:DS1820_BUS2:Temp13	13	Температура, канал 13 (BUS2, второй пин STM32)
14	M31MAGN:DS1820_BUS2:Temp14	14	Температура, канал 14 (BUS2, второй пин STM32)
15	M31MAGN:DS1820_BUS2:Temp15	15	Температура, канал 15 (BUS2, второй пин STM32)
16	M31MAGN:DS1820_BUS2:Temp16	16	Температура, канал 16 (BUS2, второй пин STM32)
17	M31MAGN:DS1820_BUS2:Temp17	17	Температура, канал 17 (BUS2, второй пин STM32)
18	M31MAGN:DS1820_BUS2:Temp18	18	Температура, канал 18 (BUS2, второй пин STM32)
19	M31MAGN:DS1820_BUS2:Temp19	19	Температура, канал 19 (BUS2, второй пин STM32)

Продолжение на следующей странице

#	PV (M31, проект)	Offset	Примечание
20	M31MAGN:DS1820_BUS2:Temp20	20	Температура, канал 20 (BUS2, второй пин STM32)
21	M31MAGN:DS1820_BUS2:Temp21	21	Температура, канал 21 (BUS2, второй пин STM32)
22	M31MAGN:DS1820_BUS2:Temp22	22	Температура, канал 22 (BUS2, второй пин STM32)
23	M31MAGN:DS1820_BUS2:Temp23	23	Температура, канал 23 (BUS2, второй пин STM32)

Примечание. В варианте B адрес регистра Modbus для BUS2 определяется как $A_2 + \texttt{Offset}$, где A_2 — стартовый адрес второго блока (например, $A_2 = 108$). Для сохранения совместимости с существующими панелями CSS возможна агрегация (вариант A) либо введение синонимов PV/переназначение каналов на уровне шаблонов.

План работ и методика испытаний

План работ по программной части

STM32: формализация карты каналов и протокола (или таблицы регистров Modbus); поддержка второй гирлянды; стендовые тесты на полном количестве датчиков.

Raspberry Pi/IOC: подготовка базы PV и шаблонов, реализация асинхронной модели опроса или адаптация к Modbus/asyn, интеграция в CSS/архивирование; тесты на тайм-ауты и устойчивость.

План работ по аппаратной части

Подготовка перечня компонентов, датчиков и кабельной обвязки, выполнение пайки/сборки второй гирлянды; проверка на стенде и последующая интеграция на объекте.

Методика проверки и критерии готовности

- проверка чтения всех каналов обеих гирлянд и сопоставление с эталонным термометром на части точек;
- измерение задержек обновления PV (периодичность, джиттер) и устойчивость при нагрузке;
- имитация отказов (обрыв/КЗ/потеря связи) и проверка корректности

статусов PV и тревог в CSS;

- проверка корректности архивирования и целостности временных рядов.

ЗАКЛЮЧЕНИЕ

В промежуточном отчёте выполнен обзор установки СПАСЧАРМ и места спектрометрического магнита М31 в составе магнитного спектрометра. На основе внутренней документации и диссертационных материалов сформулирована роль температурного контроля для водоохлаждаемых магнитов, выделены типовые аварийные сценарии и требования к диагностике.

Проанализированы предоставленные программные и схемотехнические материалы, относящиеся к измерительному модулю (STM32 + 1-Wire + Modbus RTU) и к инфраструктуре EPICS IOC на Raspberry Pi. Показано, что при расширении числа каналов существенным становится вопрос неблокирующей (асинхронной) модели ввода-вывода, обеспечивающей масштабируемость и устойчивость системы медленного контроля. Сформулирован проект расширения второй гирляндой датчиков для М31, предложены принципы именования PV, подход к формированию статусов/тревог, а также план работ и методика испытаний.

СПИСОК ЛИТЕРАТУРЫ

Список литературы

[1] Установка СПАСЧАРМ. Описание экспериментального комплекса и подсистем (внутренний документ / редакция). Файл: *Установка СПАСЧАРМ_ПТЭ_ред.docx*.

[2] Букреева С. И. Распределённая система управления детекторами эксперимента СПАСЧАРМ: диссертация (PDF). Файл: *Диссертация.pdf*.

[3] Материалы проекта EPICS IOC (конфигурация `st.cmd`, базы `.db`, шаблоны `.substitutions`), архив *ioc.zip*.

[4] Пример исходного кода прошивки микроконтроллера STM32 для опроса датчиков температуры 1-Wire и обмена по Modbus RTU. Файл: *main.c*.

[5] Схема измерительного модуля контроля температуры/охлаждения. Файл: *Temp_water_magnet_схема.PDF*.

ПРИЛОЖЕНИЯ

Фрагменты исходного кода STM32 (пример)

```
#include "stm32f2xx.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <float.h>
#include "system_stm32f2xx.c"

#include "main.h"
#include "ds.c"
#include "usart.c"
#include "gpio.c"
#include "spi.c"
#include "modbus.c"

unsigned char ds_data[30];

char cmd[80];
char rxbuf[80];
unsigned char rxmsg_rdy, rx_idx;
unsigned char Error;
unsigned int addr = 0;
unsigned long int code = 0;

unsigned char rs485;

unsigned int curr_temp[];
unsigned char rslt;

unsigned char ds_ext[];
unsigned char ds[8];

unsigned int getparams(char *str, unsigned int *addr, unsigned long int
    *code){
    char *p1,*p2;
    int i = 0;
    p1=strtok(str," ,"); /* SPACE as a delimiter */
    if (p1) {
```

```
38        i++;
39    *addr = atoi(p1);
40    p2=strtok(NULL," ,"); /* next token */
41    if (p2) {
42       i++;
43       *code = atoi(p2);
44    }
45      }
46      return i;
47  }
48
49  void RTC_init(void)
50  {
51    RCC->APB1ENR|=RCC_APB1ENR_PWREN;
52    PWR->CR|=PWR_CR_DBP;
53    RCC->CSR|=RCC_CSR_LSION;
54    while(!(RCC->CSR&RCC_CSR_LSIRDY));
55    if(RTC->ISR&RTC_ISR_INITS)return;
56    RCC->BDCR|=RCC_BDCR_BDRST;
57    RCC->BDCR&=~RCC_BDCR_BDRST;
58    RCC->BDCR&=~RCC_BDCR_RTCSEL;
59    RCC->BDCR|=(RCC_BDCR_RTCSEL_1);
60    RCC->BDCR|=RCC_BDCR_RTCEN;
61    RTC->WPR=0xCA;
62    RTC->WPR=0x53;
63    RTC->ISR|=RTC_ISR_INIT;
64    while(!(RTC->ISR&RTC_ISR_INITF));
65    RTC->PRER=249;
66    RTC->PRER=249|(127<<16);
67    RTC->CR|=RTC_CR_FMT;
68    RTC->ISR&=~RTC_ISR_INIT;
69    RTC->WPR=0;
70    RTC->WPR=0;
71  }
72
73  void TIMER_3_init(void) { //for modbus
74  /* Initializes TIM3. It is to communicate on Modbus */
75  /*                       3.5 char.
76    1.5 char              counter */
77      RCC -> APB1ENR |= RCC_APB1ENR_TIM3EN; /* Clock on TIM2 = 8 MHz */
78      TIM3 -> CNT = 0x0; /* Counter value */
```

```
79      TIM3 -> PSC = 0x18F; /* Prescaler value 399+1 */
80      TIM3 -> ARR = 0x74; /* Auto-reload value (overflow value) */
81      TIM3 -> DIER |= TIM_DIER_UIE; /* Update interrupt enabled */
82      NVIC_EnableIRQ (TIM3_IRQn);
83  }
84
85  void TIM3_IRQHandler(void){
86
87  /* Overflow every ~2,9 ms (3,5 char) on 9600bps*/
88      TIM3 -> SR &= ~TIM_SR_UIF; /* Clear update interrupt flag */
89      TIM3 -> CR1 &= ~TIM_CR1_CEN; /* Counter disabled */
90      if ((receive_message[0] == modbus_address)
91        ||(receive_message[0] == 0x00)) { /* Check address or broadcast */
92          FlagModbus = 1; /* Need to encode modbus_message */
93          modbus_char_counter = receive_char_counter - 2; /* Minus 2 bytes
              CRC */
94          for (unsigned int i=0; i < modbus_char_counter; i++)
95              modbus_message[i] = receive_message[i];
96          modbus_encoder();
97      }
98      receive_char_counter = 0;
99  }
100
101
102 void TIMER_2_init(void) { //update parameters
103     RCC -> APB1ENR |= RCC_APB1ENR_TIM2EN;
104     TIM2 -> CNT = 0x0;
105     TIM2 -> PSC = 0x7FFF;
106     TIM2 -> ARR = 0x6F0; //~3s,      ,       modbus
107     TIM2 -> DIER |= TIM_DIER_UIE;
108     NVIC_EnableIRQ (TIM2_IRQn);
109     TIM2 -> CR1 |= TIM_CR1_CEN;
110 }
111 void TIM2_IRQHandler(void){
112     TIM2 -> SR &= ~TIM_SR_UIF;
113     FlagUpdate = 1;
114 }
115
116
117
118
```

```
119
120 int main()
121 {
122 SystemInit();
123
124 GPIO_init();
125 USART1_init();//     USART1 19200,     USART3 9600  (    APB).
126
127
128 if (rs485 == 0){
129     sprintf(cmd,"Cockroft-Walton control v1.0:\n\r");
130     USART1_send_Str(cmd);
131     sprintf(cmd,"Chip STM32F205VET6, frequency 32MHz. Press H to
          help.\n\r");
132     USART1_send_Str(cmd);
133 }
134 if (rs485 == 1){
135     MODBUS_init();
136     TIMER_3_init();
137 }
138
139
140  GPIOB -> ODR |= GPIO_ODR_ODR_1; //LED
141 //////Init temp sensors
142 USART3_init();
143 DS2480B_Detect();
144 LastDiscrepancy = 0;
145 LastDeviceFlag = FALSE;
146 LastFamilyDiscrepancy = 0;
147 rslt = OWSearch();
148 num=0;
149 while (rslt) {
150   for (unsigned char u=0; u < 8; u++) ds_ext[u+num] = ROM_NO[u];
151   num = num + 8;
152   rslt = OWSearch();
153 }
154 num=num/8;
155
156 for (unsigned char u = 0; u < num; u++) {
157   for (unsigned char k=0; k < 8; k++) ds[k] = ds_ext[k+8*u];
158   Match_Write_config(ds);
```

```
159 }
160 for (unsigned char u = 0; u < 32; u++) curr_temp[u] = 0 ;
161
162 TIMER_2_init();
163 ///////
164
165
166
167 while(1) {
168   /*if (FlagModbus == 1) {      //            ,        modbus   -
                  (750ms)
169       modbus_encoder();
170   }*/
171
172   if (FlagUpdate == 1) {
173       FlagUpdate = 0;
174       for (unsigned char u = 0; u < num; u++) {
175           for (unsigned char k=0; k < 8; k++) ds[k] = ds_ext[k+8*u];
176           Match_Read_Temp(ds);
177           curr_temp[u] = ((ds_data[5]<<8) + ds_data[4]) ;
178           real_temp[u] = ((ds_data[5]<<8) + ds_data[4]) * 0.0625;
179       }
180   }
181
182
183 }
184 }
```

Конфигурация запуска IOC (st.cmd)

```
#!/home/pi/ioc/bin/linux-arm/EpicsTut

< /home/pi/ioc/iocBoot/iocEpicsTut/envPaths
epicsEnvSet("STREAM_PROTOCOL_PATH", "/home/pi/ioc/proto")

## Register all support components
dbLoadDatabase("/home/pi/ioc/dbd/EpicsTut.dbd",0,0)
EpicsTut_registerRecordDeviceDriver(pdbbase)

## Serial Port configuration
drvAsynSerialPortConfigure( "ttyAMA0", "/dev/ttyAMA0" )
asynSetOption( "ttyAMA0", 0, "baud", "9600" )
asynSetOption( "ttyAMA0", 0, "bits", "8" )
asynSetOption( "ttyAMA0", 0, "parity", "none" )
asynSetOption( "ttyAMA0", 0, "stop", "1" )
asynSetOption( "ttyAMA0", 0, "clocal", "Y" )
asynSetOption( "ttyAMA0", 0, "crtscts", "N" )
asynSetOption( "ttyAMA0", 0, "ixon", "N" )
asynSetOption( "ttyAMA0", 0, "ixoff", "N" )

## Modbus configuration
modbusInterposeConfig("ttyAMA0", 1,0, 4)



##      (                 )
##                        ,
##               .      dino.substitutions
##         1  23     (23  CSS -            ).



drvModbusAsynConfigure("RMB3_1","ttyAMA0",16, 3, 72, 2, 0, 5000, "dino")
drvModbusAsynConfigure("RMB3_2","ttyAMA0",16, 3, 84, 24, 0, 5000, "dino")
drvModbusAsynConfigure("RMB3_3","ttyAMA0",16, 3, 116, 3, 0, 5000, "dino")
drvModbusAsynConfigure("RMB3_4","ttyAMA0",16, 3, 67, 1, 0, 5000, "dino")
drvModbusAsynConfigure("WMB5_0","ttyAMA0",16, 5, 21069, 1, 0, 5000, "dino")


```

```
41 ## Load record instances
42 #dbLoadRecords("/home/pi/ioc/db/dino.db")
43 dbLoadTemplate("/home/pi/ioc/iocBoot/iocEpicsTut/dino.substitutions")
44
45 iocInit()
46
47 ## Start any sequence programs
48 #seq sncEpicsTut,"user=pi"
```

Пример базы PV для температурных каналов (dino.db, магнит M29)

```
1
2  record (ai, "DINOMAGN:AM3202:Humid"){
3    field( DTYP, "asynInt32")
4    field( INP, "@asyn(RMB3_1, 0) MODBUS_DATA" )
5    field( ASLO, "0.1")
6    field( AOFF, "0")
7  # field( HIGH, "35")
8  # field( HSV, "MINOR")
9  # field( HIHI, "45")
10 # field( HHSV, "MAJOR")
11   field( SCAN, "1 second")
12 }
13
14 record (ai, "DINOMAGN:AM3202:Temp"){
15   field( DTYP, "asynInt32")
16   field( INP, "@asyn(RMB3_1, 1) MODBUS_DATA" )
17   field( ASLO, "0.1")
18   field( AOFF, "0")
19   field( HIGH, "30")
20   field( HSV, "MINOR")
21   field( HIHI, "40")
22   field( HHSV, "MAJOR")
23   field( SCAN, "1 second")
24 }
25
26
27 record (ai, "DINOMAGN:DS1820:Amount"){
28   field( DTYP, "asynInt32")
29   field( INP, "@asyn(RMB3_2, 0) MODBUS_DATA" )
30   field( LOW, "22")
31   field( LSV, "MINOR")
32   field( LOLO, "21")
33   field( LLSV, "MAJOR")
34   field( SCAN, "1 second")
35 }
36
37 record (ai, "DINOMAGN:DS1820:Temp${CH}"){
38   field( DTYP, "asynInt32")
39   field( INP, "@asyn(RMB3_2, ${OFFSET}) MODBUS_DATA" )
40   field( ASLO, "0.1")
```

```
  field( AOFF, "0")
  field( HIGH, "55")
  field( HSV, "MINOR")
  field( HIHI, "60")
  field( HHSV, "MAJOR")
  field( SCAN, "1 second")
}

record (ai, "DINOMAGN:BME280:Humid"){
  field( DTYP, "asynInt32")
  field( INP, "@asyn(RMB3_3, 0) MODBUS_DATA" )
  field( ASLO, "0.0039")
  field( AOFF, "0")
# field( HIGH, "35")
# field( HSV, "MINOR")
# field( HIHI, "40")
# field( HHSV, "MAJOR")
  field( SCAN, "1 second")
}

record (ai, "DINOMAGN:BME280:Temp"){
  field( DTYP, "asynInt32")
  field( INP, "@asyn(RMB3_3, 1) MODBUS_DATA" )
  field( ASLO, "0.1")
  field( AOFF, "0")
  field( HIGH, "30")
  field( HSV, "MINOR")
  field( HIHI, "40")
  field( HHSV, "MAJOR")
  field( SCAN, "1 second")
}


record (ai, "DINOMAGN:BME280:Pres"){
  field( DTYP, "asynInt32")
  field( INP, "@asyn(RMB3_3, 2) MODBUS_DATA" )
  field( ASLO, "1")
  field( AOFF, "0")
# field( HIGH, "770")
# field( HSV, "MINOR")
# field( LOW, "750")
```

```
# field( LSV, "MINOR")
  field( SCAN, "1 second")
}



record (ai, "DINOMAGN:I"){
  field( DTYP, "asynInt32")
  field( INP, "@asyn(RMB3_4, 0) MODBUS_DATA" )
# field( ASLO, "0.10714285714")
  field( ASLO, "0.1")
  field( AOFF, "0")
  field( HIGH, "1500")
  field( HSV, "MAJOR")
  field( LOW, "1200")
  field( LSV, "MINOR")
  field( SCAN, "1 second")
}



record (ao, "DINOMAGN:Reset"){
  field( DTYP, "asynInt32")
  field( OUT, "@asynMask(WMB5_0,0,1,1) MODBUS_DATA" )
}
```

Шаблон базы PV для магнита M31 (m31.db, проект)

```
#       EPICS PV       31     ().
#            dino.db     ( 29),                              1-Wire (BUS2).
#    :       RMB3_*                                    st.cmd.

record (ai, "M31MAGN:DS1820:Amount"){
  field( DTYP, "asynInt32")
  field( INP, "@asyn(RMB3_2, 0) MODBUS_DATA" )
  #                                        BUS1.
  # field( LOW, "11")
  # field( LSV, "MINOR")
  # field( LOLO, "10")
  # field( LLSV, "MAJOR")
  field( SCAN, "I/O Intr")
}

record (ai, "M31MAGN:DS1820:Temp${CH}"){
  field( DTYP, "asynInt32")
  field( INP, "@asyn(RMB3_2, ${OFFSET}) MODBUS_DATA" )
  field( ASLO, "0.1")
  field( AOFF, "0")
  field( HIGH, "55")
  field( HSV, "MINOR")
  field( HIHI, "60")
  field( HHSV, "MAJOR")
  field( SCAN, "I/O Intr")
}

record (ai, "M31MAGN:DS1820_BUS2:Amount"){
  field( DTYP, "asynInt32")
  field( INP, "@asyn(RMB3_5, 0) MODBUS_DATA" )
  #                              BUS2.
  field( SCAN, "I/O Intr")
}

record (ai, "M31MAGN:DS1820_BUS2:Temp${CH}"){
  field( DTYP, "asynInt32")
  field( INP, "@asyn(RMB3_5, ${OFFSET}) MODBUS_DATA" )
  field( ASLO, "0.1")
  field( AOFF, "0")
  field( HIGH, "55")
```

```
41   field( HSV, "MINOR")
42   field( HIHI, "60")
43   field( HHSV, "MAJOR")
44   field( SCAN, "I/O Intr")
45 }
```

Device Support EPICS для чтения 1-Wire через sysfs (d1w.c, прототип)

```
1  /* ANSI C includes */
2  #include <errno.h>
3  #include <stdbool.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  /* EPICS includes */
9  #include <aiRecord.h>
10 #include <alarm.h>
11 #include <dbAccess.h>
12 #include <dbScan.h>
13 #include <devSup.h>
14 #include <errlog.h>
15 #include <epicsExport.h>
16 #include <epicsTypes.h>
17 #include <iocLog.h>
18 #include <iocsh.h>
19 #include <recGbl.h>
20 #include <shareLib.h>
21
22 /*
23  * For ai records init/read/write functions can have 3 different return
       values:
24  * 0 => calculate VAL field from RVAL and linear conversion parameters
25  * 2 => do not perform convertion (value is directly written to VAL field)
26  * -1 => error
27  */
28
29 /* initialization of record instance */
30 static long init_record( aiRecord *prec ){
31
32   /* variable definitions */
33   char filename[128];
34   char *pinfo;
35   FILE * filehandle;
36   char crcMatch[5];
37   int value = 0;
38   double temperature = 0.;
39
```

```
40   prec->pact = (epicsUInt8)true; /* disable record */
41
42   /* code for initialization */
43
44   /* Contents of INP field WITHOUT leding '@' */
45   /* prec->inp.value.instio.string */
46
47   /*filename = "/sys/devices/w1_bus_master1/";
48   strcat( filename, prec->inp.value.instio.string );
49   strcat( filename, "/w1_slave" );
50 */
51   sprintf( filename, "/sys/devices/w1_bus_master1/%s/w1_slave",
        prec->inp.value.instio.string );
52   filehandle = fopen( filename, "r" );
53   if( filehandle == NULL ) {
54      fprintf( stderr, "Could not open file %s\n", filename );
55      return -1;
56   }
57 /* 2d 00 4b 46 ff ff 02 10 19 : crc=19 YES*/
58   fscanf( filehandle, "%*x %*x %*x %*x %*x %*x %*x %*x %*x : crc=%*x %s",
        crcMatch );
59   if ( strcmp( crcMatch, "YES" ) != 0 ) {
60      fprintf( stderr, "%s: CRC did not match\n", prec->name );
61     fclose( filehandle );
62      return -1;
63   }
64   fscanf( filehandle, "%*x %*x %*x %*x %*x %*x %*x %*x %*x t=%d", &value );
65   temperature = value/1000.;
66   prec->val = temperature;
67
68   fclose( filehandle );
69   pinfo = calloc( 128, sizeof(char) );
70   memcpy( pinfo, filename, 128*sizeof(char) );
71   prec->dpvt = pinfo;
72
73   prec->udf = FALSE;
74   prec->pact = (epicsUInt8)false; /* enable record */
75
76   return 2;
77 }
78
```

```
79 /* read function for record instance */
80 static long read_ai( aiRecord *prec ){
81   FILE * filehandle;
82   char crcMatch[5];
83   int value = 0;
84   double temperature = 0.;
85
86   filehandle = fopen( (char*)prec->dpvt, "r" );
87   if( filehandle == NULL ) {
88     fprintf( stderr, "%s: Could not open file %s\n", prec->name,
         (char*)prec->dpvt );
89     return -1;
90   }
91 /* 2d 00 4b 46 ff ff 02 10 19 : crc=19 YES*/
92   fscanf( filehandle, "%*x %*x %*x %*x %*x %*x %*x %*x %*x : crc=%*x %s",
       crcMatch );
93   if ( strcmp( crcMatch, "YES" ) != 0 ) {
94     fprintf( stderr, "%s: CRC did not match\n", prec->name );
95     fclose( filehandle );
96     return -1;
97   }
98   fscanf( filehandle, "%*x %*x %*x %*x %*x %*x %*x %*x %*x t=%d", &value );
99   temperature = value/1000.;
100  prec->val = temperature;
101
102  fclose( filehandle );
103  return 2;
104 }
105
106 /*
107  * Device Support Entry Table
108  * Function pointers to the device support routines called by the records
109  */
110 struct {
111   long number; /* number of device support routines */
112   DEVSUPFUN report; /* print report */
113   DEVSUPFUN init; /* init device support */
114   DEVSUPFUN init_record; /* init record instance */
115   DEVSUPFUN ioint_info; /* get io interrupt info */
116   DEVSUPFUN read_write; /* read/write value */
```

```
117   DEVSUPFUN special_conv; /* used by ai/ao records to calculate ESLO from
          EGUF and EGUL */
118 } dev_ai_d1w = {
119   6,
120   NULL,
121   NULL,
122   init_record,
123   NULL,
124   read_ai,
125   NULL
126 };
127
128
129 void gav(void) {}
130
131 /* export address of Device Support Entry Table to EPICS IOC */
132 epicsExportAddress( dset, dev_ai_d1w );
```