ITP Engineering
ENGINEER EVERYTHING

Семейство ПЛК Ai-Talap 241
Руководство программиста

Версия документа: 1.0
Дата создания: 13.04.2026

IT-P ENGINEERING
Республика Казахстан, г. Алматы, ул. Макатаева 129

БИН 201240027825
Қазақстан Республикасы, Алматы қаласы
Республика Казахстан, город Алматы
E-mail: info@ITPEngineering.com

ТОО «IT-P Engineering»

Содержание

1 Назначение и условия применения ПО

Семейство модульных ПЛК Ai-Talap предназначено для создания распределенных систем управления (автоматизации технологического процесса) с использованием сетей Ethernet и RS-485. ПО Ai-Talap включает следующие программы:

встроенное ПО: каждый модуль ПЛК Ai-Talap работает под управлением встроенного ПО, название которого совпадает с артикулом модуля (например, «Встроенное ПО AT241BC»);

ПО управления: программы, работающие на управляющем компьютере, обеспечивающие настройку ПЛК Ai-Talap на конкретное применение и контроль работы системы управления.

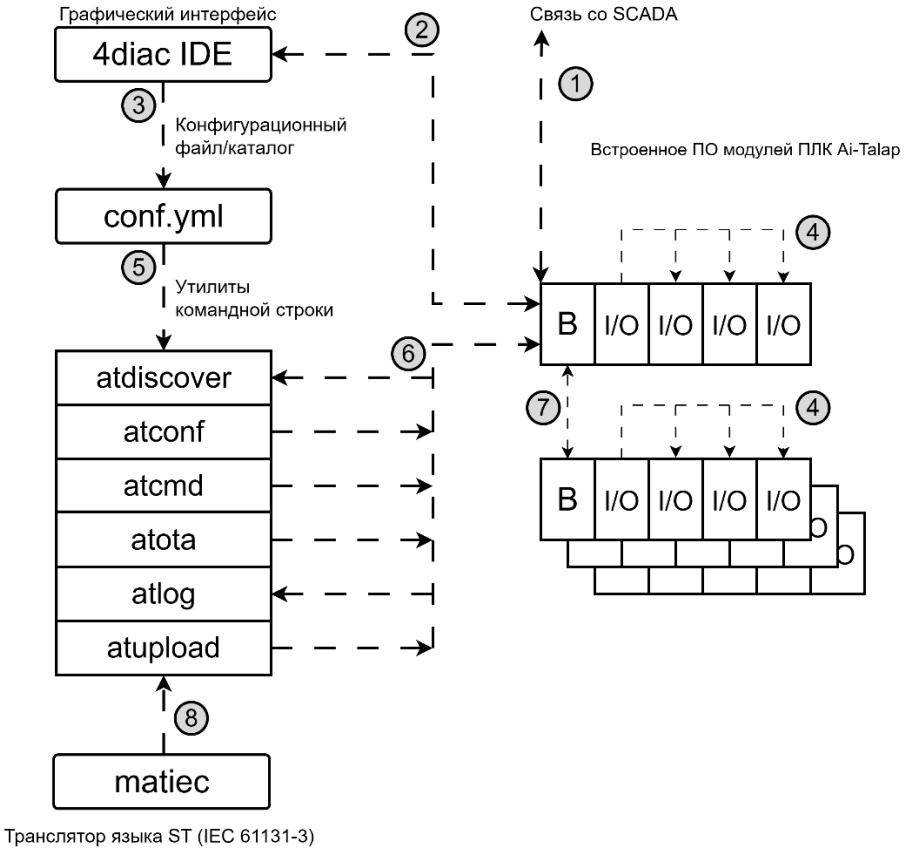

Рисунок 1.1 – Информационные связи компонентов ПО Ai-Talap. Стрелки указывают основное направление передачи информации

1.1 Назначение и взаимодействие компонентов ПО Ai-Talap

1.1.1 Встроенное ПО базового модуля

Каждый ПЛК содержит как минимум один базовый модуль (B на рис. 1.1), например AT241BC или AT241CPU, который координирует работу модулей ввода-вывода (I/O). Встроенное ПО базового модуля получает команды управления от управляющего компьютера по протоколу NPC21_Modbus (стрелка 6 на рис. 1.1); в первую очередь это команда загрузки конфигурации из файла *conf.yml* утилитой *atconf*. Конфигурация определяет режимы работы подсистем базового модуля и модулей ввода-вывода. Получив конфигурацию, ПО базового модуля загружает фрагменты конфигурации в модули ввода вывода, а затем в непрерывном режиме выполняет следующие функции:

- прием данных от модулей ввода-вывода по шине ПЛК (стрелка 4);

- прием и передача данных согласно загруженной конфигурации в SCADA
- (стрелка 1) и другие ПЛК (стрелка 7) по протоколам Modbus и IEC-104;
- передача данных в прикладные программы, выполняющиеся на ПЛК
- согласно конфигурации: программы на языке ST (IEC 61131-3) и программы на языке FBD (IEC 61499). Проекты IEC 61499 могут выполняться в ПЛК Ai-Talap одновременно с программами стандарта IEC 61131-3, взаимодействуя с последними через общее пространство именованных переменных;
- прием данных от прикладных программ;
- передача данных в модули ввода-вывода по шине ПЛК (стрелка 4) согласно конфигурации;
- передача данных на управляющий компьютер по запросу утилиты *atcmd*
- в целях контроля работы ПЛК (отладочный интерфейс);
- прием и обработка запросов от управляющего компьютера по протоколу
- NPC21_Modbus (стрелка 6), генерируемых утилитами *atconf, atcmd* и т.д.;
- прием и обработка запросов по протоколу IEC 61499 от управляющего компьютера, генерируемых графическим интерфейсом 4diac IDE (стрелка 2);
- прием и обработка данных по протоколу IEC 61499 от других ПЛК (стрелка 7).

1.1.2 Встроенное ПО модулей ввода-вывода

Встроенное ПО модулей ввода-вывода в непрерывном режиме транслирует данные между шиной ПЛК и портами ввода-вывода в соответствии с загруженной конфигурацией. В некоторых модулях (в частности, AT241AI4, AT241AO4) над данными выполняются нормирующие преобразования и применяются калибровочные поправки. Все модули поддерживают прием и обработку запросов протокола NPC21_Modbus от утилит *atconf, atcmd, atlog*; остальные утилиты не работают с модулями ввода-вывода, а только с базовыми модулями.

1.1.3 Конфигурационный файл *conf.yml*

Конфигурационный файл содержит описание параметров и режимов работы каждого модуля каждого ПЛК в распределенной системе управления в стандартном формате YAML [2]. Конфигурационный файл используется всеми утилитами (*atconf … atupload*). Редактирование файла возможно в графическом интерфейсе 4diac IDE, или в любом текстовом редакторе, желательно имеющем поддержку синтаксиса YAML (например: *emacs* или *vim*).

1.1.4 Программа *atconf*

Программа *atconf* выбирает из файла *conf.yml* и загружает в указанный ПЛК относящийся к данному ПЛК фрагмент конфигурации, используя протокол NPC21_Modbus (стрелка 6).

1.1.5 Программа *atcmd*

Программа *atcmd* позволяет выполнять команды управления на ПЛК Ai-Talap, в частности устанавливать и считывать текущие значения переменных из памяти ПЛК в целях отладки.

1.1.6 Программа *atdiscover*

Программа *atdiscover* использует *multicast ping* IPv6 для обнаружения устройств Ai-Talap в локальной сети Ethernet, в том числе устройств, ещё не имеющих конфигурации. Применяется при добавлении новых ПЛК в систему.

1.1.7 Программа *atota*

Программа *atota* выполняет удаленное (т.н. *«over the air»*) обновление встроенного ПО указанного модуля ПЛК Ai-Talap.

1.1.8 Программа *atlog*

Программа *atlog* считывает содержимое журнала системных сообщений указанного модуля ПЛК Ai-Talap в целях расширенной диагностики.

1.1.9 Программа *atupload*

Программа atupload выполняет компиляцию программ на языке ST IEC 61131-3 и загрузку исполняемого кода в ПЛК Ai-Talap. При компиляции программ программа *atupload* использует транслятор *matiec* [3] (стрелка 8), программные средства *«GNU Toolchain»* из комплекта *SDK esp-idf* [4] версии 5.3.2 и утилиту *make* из используемого дистрибутива ОС GNU/Linux.

1.1.10 Интегрированная среда разработки 4diac IDE

Среда 4diac IDE [5] представляет собой свободное программное обеспечение (лицензия Eclipse Public License v2.0) для программирования распределенных систем промышленной автоматизации в соответствии со стандартом IEC 61499 в графической форме диаграмм функциональных блоков (FBD). Среда 4diac IDE взаимодействует по протоколу TCP (порт 61499) с компонентой 4diac FORTE, встроенной в ПО базовых модулей ПЛК Ai-Talap, обеспечивающей загрузку и выполнение проектов IEC 61499 в ПЛК Ai-Talap.

Экземпляры 4diac FORTE в разных ПЛК, входящих в один распределенный проект 4diac, взаимодействуют по протоколам TCP/IP; конкретный протокол и порт определяются использованными в проекте коммуникационными функциональными блоками. Для изучения практической реализации программирования в 4diac необходимо обратиться к разделу 8.

1.2 Условия применения ПО

Встроенное ПО Ai-Talap работает модулях ПЛК Ai-Talap соответствующих типов без дополнительных условий.

ПО управления работает в среде ОС Linux на компьютере с архитектурой AMD64, требует не менее 4ГБ оперативной памяти и не менее 10ГБ дискового пространства. При разработке и тестировании используется дистрибутив *Debian GNU/Linux 12 (bookworm)*; возможна адаптация ПО управления под любой современный дистрибутив ОС Linux.

ПО управления взаимодействует с ПЛК Ai-Talap в локальной сети Ethernet по протоколу Modbus TCP. Для упрощения конфигурации ПЛК желательно наличие сервера DHCP в локальной сети, хотя возможно и статическое назначение адреса. Если в конфигурации ПЛК задан статический адрес IPv4, утилиты ПО управления используют Modbus TCP IPv4, иначе Modbus TCP IPv6.

2 Характеристики программы

2.1 Принципы организации ПО Ai-Talap

ПЛК Ai-Talap позволяют одновременно (в режиме разделения времени) исполнять одну или несколько прикладных программ, написанных на языках стандарта IEC 61131-3 [6], реализующих логику управления. В текущей версии ПО поддерживается только один из четырех определенных в стандарте языков – язык ST.

Программы ПЛК поддерживают связь с объектом управления через входные и выходные переменные языка ST. Переменные соответствуют сигналам на интерфейсах модулей ПЛК, или данным удаленных устройств ввода-вывода, взаимодействующих с ПЛК по протоколам Modbus [7] или IEC 60870-5-104 [8].

2.1.1 Брокер и конфигурация

Центральным компонентом встроенного ПО ПЛК Ai-Talap является брокер данных, реализующий взаимодействие подсистем ПЛК по модели «издатель–подписчик». Под «подсистемами» имеются в виду как аппаратные компоненты (модули расширения, установленные в слотах шины ПЛК), так и программные, например *modbus_server*, *modbus_client, plcprog* (подсистема исполнения программ ПЛК). Когда одна подсистема (выступающая в этот момент как «издатель») вырабатывает новое значение переменной, брокер сохраняет обновленное значение для выдачи по запросам, а также пересылает новое значение подсистемам-подписчикам данной переменной. Каждая подсистема может быть издателем и подписчиком для произвольного числа переменных; конкретные связи «издатель-подписчики» устанавливаются (изменяются) в момент начальной (или повторной) загрузки конфигурации в ПЛК.

Конфигурация всех ПЛК, входящих в состав распределенной системы управления, описывается в формате YAML [2] в файле *conf.yml* и загружается в ПЛК командой *atconf* (раздел 3.2). Конфигурация задает аппаратный состав ПЛК (установленные модули), режимы каналов ввода-вывода и настройку программных подсистем; директивы конфигурации описаны в разделе 5.

Описание конфигурации имеет иерархическую структуру, в которой каждой подсистеме и ее блокам (например, каждому каналу 4-канального модуля аналогового ввода) соответствует свой подраздел (поддерево иерархии). Для указания, что данный блок является «издателем» (источником) переменной name, в его подразделе указывается атрибут *var: name*. Для указания, что данный блок является «подписчиком» (получателем) переменной name, в его подразделе указывается атрибут *var_out: name*. У каждой переменной может быть только один издатель, и неограниченное число подписчиков. Исключением являются переменные с атрибутом multisource, у которых может быть несколько издателей; при использовании таких переменных прикладной программист должен обеспечить, чтобы интервал между изменением переменной разными издателями был не меньше времени, необходимого для распространения изменений всем подписчикам.

Область видимости переменных ограничивается одним ПЛК, т. е. одно имя переменной в разных ПЛК можно использовать для разных целей, фактически это разные переменные.

2.1.2 Прикладные программы на языке ST (IEC 61131-3)

Программы ПЛК на языке ST IEC 61131-3 создаются на управляющем компьютере с помощью текстового редактора и описываются в конфигурации (разделы *plcN: taskN: progN*): указывается имя файла с программой и параметры ее выполнения. Загрузка программ ПЛК реализована отдельно от загрузки конфигурации и выполняется программой *atupload* (раздел 3.6). Для запуска программы необходимо:

- чтобы программа была загружена;

➢ чтобы исполнение данной программы было разрешено в конфигурации (*plcN: taskN: progN: enable: y*);

➢ чтобы был установлен общий флаг разрешения выполнения программ: *sys.run=1* (раздел 3.3.5).

Привязка программ ПЛК к физическим интерфейсам ПЛК осуществляется через переменные брокера. Как входные, так и выходные переменные должны быть описаны в программе на языке ST как *VAR_EXTERNAL*, а также перечислены в разделе var конфигурации (см. 5.2.6); для входные переменных должен быть указан атрибут *init*.

Выходные переменные программ ПЛК обновляются часто (параметр *task:period*, раздел *refsec:conf.yml-task*), и поэтому обрабатываются брокером особо: если при обновлении не произошло изменения значения переменной, брокер не рассылает переменную подписчикам. Можно разрешить рассылку, задав для переменной атрибут *interval* в разделе var конфигурации (см. 5.2.6).

2.1.3 Прикладные программы на языке FBD (IEC 61499)

Программы ПЛК на языке FBD IEC 61499 (также называемые «проектами 4diac») разрабатываются в интегрированной среде 4diac IDE [5]. Среда позволяет загрузить *(«Развернуть», «Deploy»)* проект в указанный ПЛК и наблюдать за его работой (изменениями значений переменных во времени). Проект исполняется на ПЛК программной компонентой 4diac FORTE, входящей в состав встроенного ПО модулей AT241CPU.

Привязка проектов 4diac к физическим интерфейсам ПЛК осуществляется через переменные брокера, перечисленные в разделе forte конфигурации (см. 5.2.2). В проектах 4diac переменные из списка *forte:var* необходимо привязать к функциональным блокам *PUBLISH*, а переменные *forte:var_out* к функциональным блокам *SUBSCRIBE*.

Для автоматической загрузки проекта forte на выполнение после перезагрузки ПЛК необходимо в 4diac IDE сгенерировать файл *forte.fboot* и загрузить его в постоянную память ПЛК (раздел 3.3.8).

2.1.4 Передача данных между ПЛК

Для передачи данных между ПЛК в распределенной системе управления используются протоколы Modbus или IEC-104; при этом на передающем ПЛК передаваемая переменная должна быть описана как *var_out* в подсистеме Modbus (или IEC-104), а на принимающем ПЛК должна быть соответствующая входная переменная, описанная как *var* в подсистеме Modbus (или IEC-104).

Имена переменных на передающем и принимающем ПЛК могут совпадать или различаться; соответствие между переменными устанавливается по адресам регистров Modbus (или объектов IEC-104).

2.1.5 Интерфейс управления и отладки

Для контроля и управления работой ПЛК реализован интерфейс управления и отладки, представленный для пользователя командой *atcmd* (раздел 3.3). Интерфейс позволяет выводить значения переменных и устанавливать их, а также выполнять команды управления. Реализация базируется на передаче JSON-объектов с командами и данными по протоколу NPC21_Modbus (Modbus TCP или Modbus RTU, дополнительная функция fn=107) и допускает неограниченное расширение набора команд.

3 Обращение к программам

3.1 Обращение к программе *atdiscover*

Программа *atdiscover* выполняет обнаружение ПЛК Ai-Talap 241 в локальной сети, непосредственно подключенной к компьютеру. Программа обнаруживает все ПЛК, как имеющие, так и не имеющие конфигурации. При запуске без аргументов программа *atdiscover* выводит информацию обо всех ПЛК Ai-Talap 241, присутствующих в локальной сети:

```
$ atdiscover
plc1 AT241CPU [192.168.2.1] f4:12:fa:dd:c6:17 eth0
plc102 AT241CPU [192.168.2.102] f4:12:fa:dd:c5:e3 eth0
<noname> AT241CPU [192.168.1.1] f4:12:fa:dd:c6:e4 eth1
```

Каждая строка в выдаче программы описывает один обнаруженный ПЛК:

- *plc1*: имя ПЛК согласно конфигурации, загруженной в ПЛК на данный момент. Если в ПЛК еще не загружена конфигурация, вместо имени ПЛК выводится <*noname*>.
- *192.168.2.1*: адрес IPv4, выданный ПЛК сервером DHCP или заданный статически при конфигурации ПЛК. Это сопутствующая информация, не имеющая большого значения: при обращениях к ПЛК по имени все утилиты определяют адрес автоматически по конфигурационному файлу. При изменении адреса ПЛК (выдачи нового адреса сервером DHCP или задания нового статического адреса в конфигурации) новый адрес автоматически регистрируется в системе mDNS. Если на компьютере настроено использование mDNS (в /etc/nsswitch.conf), можно обращаться к ПЛК не по IP-адресу, а по имени, например: *ping plc102.local*.
- *f4:12:fa:dd:c6:17*: адрес уровня 2 модели OSI (MAC-адрес Ethernet, EUI-48). Постоянный уникальный идентификатор данного ПЛК. Получение MAC-адресов ПЛК, не имеющих конфигурации, является главной целью программы discover. Адрес необходим для однозначной идентификации ПЛК при задании его конфигурации в файле *conf.yml*, он должен быть задан в обязательном параметре *macaddr: plcN: {macaddr: f4:12:fa:d4:d1:7f}*.
- *eth0*: интерфейс локальной сети, в которой обнаружен ПЛК – на случай, когда управляющий компьютер имеет несколько интерфейсов локальной сети. Это сопутствующая информация, не имеющая большого значения: при обращениях к ПЛК по имени все утилиты определяют интерфейс автоматически по конфигурационному файлу.

3.1.1 Опциональные аргументы программы *atdiscover*

- *--help*: команда *atdiscover* выдает краткую справку по использованию и завершается:

```
$ atdiscover --help
Usage: atdiscover [-n|--newonly] [-f /path/to/conf.yml]
```

- *-f /path/to/conf.yml*: задает альтернативный путь к файлу конфигурации. По умолчанию программа *atdiscover* использует файл конфигурации *%BASEDIR%/conf/conf.yml*, который находится исходя из пути запуска программы *atdiscover* (*%BASEDIR%/bin/atconf*);
- *-n*: опция подавляет вывод информации о ПЛК, уже присутствующих в файле конфигурации, и удобна для обнаружения новых ПЛК, требующих конфигурации, при большом числе ПЛК в локальной сети.

3.2 Обращение к программе *atconf*

Программа *atconf* выполняет загрузку конфигурации ПЛК Ai-Talap 241. Конфигурация всех ПЛК, входящих в состав распределенной системы управления, задается в файле *conf.yml*. Формат файла конфигурации описан в разделе 5.

При запуске программы *atconf* с указанием имени ПЛК в качестве аргумента программа выделяет из файла *conf.yml* раздел конфигурации, относящийся к данному ПЛК, сериализует в формате JSON и передает на ПЛК по протоколу Modbus TCP:

```
$ atconf plc1
Configuration status: OK
```

Между запуском программы *atconf* и выдачей сообщения "*Configuration status: OK"* или сообщения об ошибке обычно проходит 2-5 секунд (не более 10).

3.2.1 Опциональные аргументы программы *atconf*

- *--help*: команда *atconf* выдает краткую справку по использованию и завершается:

```
$ atconf --help
Usage: atconf {PLCNAME|--all} [-f /path/to/conf.yml]  [--bymac]
```

- *-f /path/to/conf.yml*: задает альтернативный путь к файлу конфигурации. По умолчанию программа atconf использует файл конфигурации *%BASEDIR%/conf/conf.yml*, который находится исходя из пути запуска програмы *atconf (%BASEDIR%/bin/atconf)*;
- *--all*: аргумент можно указать вместо имени ПЛК, если требуется обновить конфигурацию всех ПЛК, описанных в файле *conf.yml*.
- *--bymac*: устанавливает режим обращения к ПЛК по локальному (*link local*) адресу IPv6, даже если для целевого ПЛК в файле *conf.yml* задан статический адрес IPv4 (смотреть раздел 3.2.2).

3.2.2 Замечания по реализации

Программа *atconf* обращается к ПЛК по протоколу Modbus TCP с использованием дополнительной (*user-defined*) функции *fn = 107*. По умолчанию обращение производится по локальному (*link local, fe80::/10*) адресу IPv6, сформированному на основе MAC-адреса, заданного обязательным параметром *macaddr* в конфигурационном файле *conf.yml*.

В случае, когда в конфигурационном файле *conf.yml* для целевого ПЛК задан статический адрес IPv4 (параметр *ipaddr*), обращение производится по этому адресу – это полезно в случае, если ПЛК находится не в одной локальной сети с управляющим компьютером. Можно потребовать использовать локальный адрес IPv6 с помощью параметра *--bymac* (смотреть раздел 3.2.1).

В ПЛК Ai-Talap сервер Modbus TCP IPv6 на стандартном порту 502 включен всегда; раздел *modbus_server* в файле *conf.yml* управляет только серверами Modbus TCP IPv4 и Modbus RTU. Таким образом, возможность обновления конфигурации ПЛК, находящегося в одной локальной сети с управляющим компьютером, сохраняется всегда, даже при неверных параметрах TCP/IP и при изменении параметра *modbus_server:tcp_port* в файле *conf.yml*.

3.3 Обращение к программе *atcmd*

Программа *atcmd* позволяет выполнять на ПЛК команды считывания и установки переменных в целях отладки, а также некоторые другие команды.

Программа *atcmd* требует один обязательный аргумент: имя ПЛК из конфигурационного файла *conf.yml*, например, *plc1*. Дополнительные аргументы варьируются в зависимости от выполняемой команды.

3.3.1 Команда «считывание переменных»

Следующая команда запрашивает у ПЛК с именем *plc1* значения четырех переменных и выводит имена переменных и их значения на стандартный вывод:

```
$ atcmd plc1 di0..2 motor_speed
2025-10-25 15:18:47 di0=0 di1=0 di2=1 motor_speed=97
```

Программа *atcmd* не имеет ограничения на количество переменных в запросе, но в текущей реализации встроенного ПО ПЛК есть ограничение на размер ответа ПЛК в формате JSON: 218 байтов. Таким образом, ограничение на количество переменных в запросе зависит от суммы длин имен переменных и их значений в текстовом виде.

3.3.2 Команда «установка значений переменных»

Следующая команда выполняет установку на ПЛК с именем *plc1* значений двух переменных: переменная *do0* получает значение 1, переменная с именем *motor_speed* получает значение 50. При отсутствии ошибок команда не выдает сообщений и завершается с кодом 0.

```
$ atcmd plc1 do0=1 motor_speed=50
```

3.3.3 Команда «установка значений переменных в режиме force»

Приведенная ниже команда выполняет установку на ПЛК с именем *plc1* значений двух переменных: переменная *do0* получает значение 1, переменная с именем *motor_speed* получает значение 50. Ключ *-F* задает перевод переменных в «принудительный» (*«force»*) режим, в котором встроенное ПО ПЛК игнорирует попытки изменения этих переменных со стороны программ ПЛК или модулей ввода-вывода. Режим *«force»* сохраняется до его отмены с помощью команды *cmd -U*, или до перезагрузки ПЛК.

При отсутствии ошибок команда не выдает сообщений и завершается с кодом 0.

```
$ atcmd plc1 -F do0=1 motor_speed=50
```

3.3.4 Команда «отмена режима force»

Следующая команда отменяет на ПЛК с именем *plc1* ранее установленный режим *«force»* для переменных с именами *do0* и *motor_speed*. Если указанные переменные не находились в режиме *«force»*, команда не имеет эффекта, но не считает эту ситуацию ошибочной. При отсутствии ошибок команда не выдает сообщений и завершается с кодом 0.

```
$ atcmd plc1 -U do0=1 motor_speed=50
```

3.3.5 Системные переменные

Имена вида *sys.XXX* зарезервированы для системных переменных, позволяющих получить информацию об указанном модуле ПЛК или изменить режим его работы. Сводный список системных переменных приведен в таблице 3.1; конкретный набор переменных свой у каждого типа модуля.

Таблица 3.1 – Системные переменные

Имя переменной	Режим доступа	Описание
sys.alarm	RO	Биты ошибок головного модуля
sys.alarmN	RO	Биты ошибок модуля номер N
sys.btnage	RO	Время в секундах с момента нажатия на кнопку USR на лицевой панели базового модуля. Если с момента последней перезагрузки модуля кнопку не нажимали, btnage=-1
sys.btstamp	RO	Отметка времени последнего нажатия на кнопку USR на лицевой панели базового модуля (сохраненное в момент нажатия значение переменной sys.us). Если с момента последней перезагрузки модуля кнопку не нажимали, btnstamp=0
sys.bustab	WO	При записи в переменную базовый модуль выводит в лог дамп таблицы модулей в сборке (порядковый номер, тип, mac-адрес модуля). См. atlog
sys.day	RO	Текущий день (1..31): DD в компонентном представлении времени YYYY-MM-DD hh:mm:ss в локальной временной зоне
sys.devtype	RO	Тип (артикул) модуля, например AT241BC
sys.hour	RO	Текущий час (0..23): mm в компонентном представлении времени YYYY-MM-DD hh:mm:ss в локальнойвременной зоне
sys.ipaddr	RO	Текущий адрес IPv4 модуля, полученный по DHCP или заданный статически в conf.yml.
sys.macaddr	RO	MAC-адрес (уникальный идентификатор EUI-48) модуля
sys.minute	RO	Текущая минута (0..59): mm в компонентном представлении времени YYYY-MM-DD hh:mm:ss в локальной временной зоне
sys.month	RO	Текущий месяц (1..12): MM в компонентном представлении времени YYYY-MM-DD hh:mm:ss в локальной временной зоне
sys.restart	RW	Установка таймера перезагрузки (в секундах от момента установки). Значение 0 выключает таймер (отменяет ранее заказанную перезагрузку). При чтении считывается оставшееся время до перезагрузки, или 0, если таймер выключен
sys.run	RW	Разрешение исполнения программ plcprog (1=разрешено, 0=запрещено)
sys.second	RO	Текущая секунда (0..59): ss в компонентном представлении времени YYYY-MM-DD hh:mm:ss в локальной временной зоне
sys.serial	RO	Серийный номер модуля
sys.settime	WO	Запись ненулевого значения вызывает установку системного времени ПЛК (УСО) из переменной sys.wtime, с последущим обнулением sys.wtime
sys.status	RO	Биты состояния головного модуля
sys.statusN	RO	Биты состояния модуля номер N
sys.time	RO	Системное время (значение в секундах с 00:00 1.01.1970 GMT). См. также sys.wtime
sys.trace	RW	Задание режима низкоуровневой трассировки (для разработчика)
sys.uptime	RO	Время с момента последней перезагрузки модуля, секунды
sys.us	RO	Монотонный счетчик времени с момента последнего включения модуля с разрешением 1 мкс. При перезагрузке модуля без снятия питания счетчик не сбрасывается.
sys.version	RO	Версия встроенного ПО модуля

Продолжение таблицы 3.1

Имя переменной	Режим доступа	Описание
sys.wday	RO	Текущий день недели (1..7), 7 соответствует воскресенью
sys.wtime	RW	Установка системного времени (значение в секундах с 00:00 1.01.1970 GMT). Установка времени выполняется в два шага: 1) запись sys.wtime 2) запись ненулевого значения в sys.settime. Возможно чтение sys.wtime для контроля подготовленного к установке значения. После установки значение переменной sys.wtime автоматически сбрасывается в 0. См. также sys.time
sys.wdt_pause	WO	Запрет рассылки уведомлений от базовового модуля сторожевым таймерам подчиненных модулей на указанный период (мс), чтобы вызвать аппаратный сброс модуля, с которым потеряна связь
sys.yday	RO	Текущий день года (1..366)
sys.year	RO	Текущий год: YYYY в компонентном представлении времени YYYY-MM-DD hh:mm:ss в локальной временной зоне

3.3.6 Команда *calibrate*

Команда *atcmd PLCNAME/SLOT --calibrate* запускает процедуру калибровки для модулей, предусматривающих возможность калибровки. Команда не предназначена для конечного пользователя; процедура калибровки проводится на заводе-изготовителе.

3.3.7 Команда *coldrestart*

Команда *atcmd PLCNAME/SLOT --coldrestart 1* устанавливает таймер перезагрузки аналогично команде *restart* (раздел 3.3.14), но дополнительно устанавливает флаг, запрещающий загрузку retain-переменных; все переменые получают начальные значения, заданные атрибутом *init*, а при его отсутствии – значение 0.

3.3.8 Команда *fboot*

Команда *atcmd PLCNAME --fboot forte.fboot* загружает в постоянную память базового модуля ПЛК с именем *plc1* файл *forte.fboot* с проектом, подготовленным в 4diac IDE. ПЛК начнет выполнять проект в подсистеме 4diac FORTE после перезагрузки или выключения-включения питания. Перезагрузку отдельного модуля можно выполнить удаленно с помощью команды *restart* (раздел 3.3.14).

3.3.9 Команда *fsformat*

Команда *atcmd PLCNAME --fsformat* заново размечает файловую систему ПЛК, в которой хранятся файлы конфигурации, файлы программ (см. *atupload*) и файл *forte.fboot*. Фактически команда *fsformat* выполняет сброс ПЛК к заводским настройкам. После команды *–fsformat* имеет смысл сразу (до перезагрузки) заново загрузить конфигурацию (*atconf*), опционально файлы программ и файл *forte.fboot*.

3.3.10 Команда *fscheck*

Команда *atcmd PLCNAME --fscheck* проверяет корректность файловой системы, выводит в лог информацию о свободном пространстве и список файлов. См. *atlog* (раздел 3.5).

3.3,11 Команда *getconf*

Команда *atcmd PLCNAME --getconf* считывает и выводит на стандартный вывод конфигурацию, загруженную на данный момент в ПЛК с именем *PLCNAME*, в формате YAML.

3.3.12 Команда *json*

Команда *atcmd PLCNAME --json* выводит на стандартный вывод в формате JSON конфигурацию ПЛК с именем *PLCNAME*, загруженную из файла или каталога *conf.yml* (раздел 5). Команда не предназначена для конечного пользователя.

3.3.13 Команда *nofboot*

Команда *atcmd PLCNAME --nofboot* удаляет из постоянной памяти ПЛК проект 4diac, ранее загруженный командой *fboot*.

3.3.14 Команда *restart*

Команда *atcmd PLCNAME/SLOT --restart 10s* устанавливает на модуле в слоте *SLOT* ПЛК с именем *PLCNAME* таймер перезагрузки: модуль перезагрузится через 10 секунд. В аргументе команды допускаются обычные суффиксы времени: *ms, s, m, h, d*; при отсутствии суффикса предполагается *s* (секунды).

Повторный вызов команды *--restart* до срабатывания таймера обновляет счетчик таймера, т.е. откладывает перезагрузку. Периодический вызов команды *--restart* позволяет получить режим «сторожевого таймера»: ПЛК перезагрузится, если ни одна команда *restart* не поступила в течение заданного интервала.

Команда *atcmd PLCNAME/SLOT --restart 0* останавливает ранее запущенный таймер, а если таймер не был запущен, ничего не изменяет:

Значения задержки ⩽0.1s обрабатываются особо: в случае ошибки таймерной службы ПЛК, перезагрузка выполняется немедленно, без использования таймера.

Переменная *sys.restart* (таблица 1) является альтернативным каналом управления таймером перезагрузки.

3.3.15 Команда *settime*

Команда *atcmd PLCNAME --settime* устанавливает системное время ПЛК с именем *PLCNAME* равным системному времени управляющего компьютера. Время устанавливается с точностью до задержки передачи команды с компьютера на ПЛК.

Предпочтительным способом установки времени ПЛК является настройка синхронизации времени по протоколу SNTP (раздел 5.2.1), поскольку в этом случае время не только однократно устанавливается, но его точность поддерживается периодической синхронизацией.

3.3.16 Команда *stats*

Команда *atcmd PLCNAME/SLOT --stats* выводит счетчики событий, накопленные модулем в слоте *SLOT* ПЛК *PLCNAME* со времени последней перезагрузки ПЛК. Значения счетчиков могут быть полезны для диагностики. У каждого типа модуля свой набор счетчиков, по которым можно судить о наличии и частоте ошибок в работе модуля или об интенсивности использования определенных частей кода встроенного ПО модуля. Выводятся только счетчики, имеющие ненулевые значения.

3.3.17 Опциональные аргументы программы *atcmd*

- *--help*: программа *atcmd* выдает краткую справку по использованию и завершается:

```
$ atcmd --help
Usage: atcmd PLCNAME[/SLOT] [SECONDS] [-x|--hex] [-b|--bin] VAR [VAR...]
 atcmd PLCNAME[/SLOT] [SECONDS] VAR=VAL [VAR=VAL...]
 atcmd PLCNAME[/SLOT] [SECONDS] [-F|--force] VAR=VAL [VAR=VAL...]
 atcmd PLCNAME[/SLOT] [SECONDS] [-U|--unforce] VAR [VAR...]
 atcmd PLCNAME[/SLOT] --calibrate SPEC
```

```
atcmd PLCNAME[/SLOT] --fboot FILE
atcmd PLCNAME[/SLOT] --fsformat
atcmd PLCNAME[/SLOT] --fscheck
atcmd PLCNAME[/SLOT] --getconf
atcmd PLCNAME[/SLOT] --json
atcmd PLCNAME[/SLOT] --nofboot
atcmd PLCNAME[/SLOT] --restart {10s|1h|...}
atcmd PLCNAME[/SLOT] --coldrestart {10s|1h|...}
atcmd PLCNAME[/SLOT] -- settime[=SECONDS]
atcmd PLCNAME[/SLOT] --stats
atcmd PLCNAME[/SLOT] [-v] [-f conffile.yml] [--bymac [-i ethN]]...
```

- *-f /path/to/conf.yml*: задает альтернативный путь к файлу конфигурации. По умолчанию программа *atcmd* использует файл конфигурации *%BASEDIR%/conf/conf.yml*, который находится исходя из пути запуска программы *atcmd (%BASEDIR%/bin/atconf)*;
- *--bymac*: указывает, что обращение программы *atcmd* к ПЛК должно быть выполнено по локальному (link local) адресу IPv6, даже если для целевого ПЛК в файле *conf.yml* задан статический адрес IPv4 (смотреть раздел 3.2.2);
- *SECONDS*: задает интервал повторения команды и может быть любым действительным числом, например 10 или 0.5. При отсутствии аргумента SECONDS команда исполняется однократно и завершается; при наличии аргумента SECONDS она выполняется до нажатия Ctrl-C пользователем.

3.4 Обращение к программе *atota*

Программа *atota* выполняет загрузку новой версии встроенного ПО в указанный модуль ПЛК Ai-Talap, используя протокол управления NPC21_Modbus.

```
$ atota PLCNAME[/SLOT[/ADDR]] [опции] /path/to/firmware.bin[.xz]
```

Аргумент *PLCNAME/SLOT* задает имя ПЛК и номер слота модуля, в который будет загружено ПО. Имя ПЛК должно присутствовать в конфигурационном файле *conf.yml*, описание слота может отсутствовать. Если опциональная часть *[/SLOT]* не указана, обращение происходит к головному (*master*) модулю указанного ПЛК. Опциональная часть */ADDR* применяется для обращения к устройству, подключенному к внешней шине Modbus модуля AT241MG.

Файлы для загрузки публикуются изготовителем в составе ПО Ai-Talap [9] в каталоге *Ai-Talap/firmware*.

3.4.1 Опциональные аргументы программы *atota*

- *--help*: программа atota выдает краткую справку по использованию и завершается;
- *-f /path/to/conf.yml*: задает альтернативный путь к файлу конфигурации. По умолчанию программа atota использует файл конфигурации *%BASEDIR%/conf/conf.yml*, который находится исходя из пути запуска програмы *atota (%BASEDIR%/bin/atconf)*;
- *--rewind*: опция позволяет начать загрузку файла сначала. Если опция *--rewind* не указана, загрузка продолжается с места, где она была остановлена оператором (Ctrl-C) или из-за перебоев в связи, если размер загружаемого файла одинаковый в текущем и прошлом обращении к программе *atota*.

3.4.2 Примеры обращения

Загрузка ПО в модуль аналогового ввода AT241AI4, находящегося в позиции 3 в сборке УСО/ПЛК AT241, присутствующего в конфигурационном файле *conf.yml* с именем *AT1*:

```
$ atota AT1/3 ~/Ai-Talap/firmware/AT241AI4
```

3.5 Обращение к программе *atlog*

Программа *atlog* позволяет считывать содержимое журнала системных сообщений указанного модуля ПЛК Ai-Talap в целях расширенной диагностики. Журнал организован как FIFO-буфер текстовых строк; объем журнала зависит от типа модуля и варьируется от 20КБ до 2МБ. Поддерживается как считывание журнала целиком, так и выборки строк.

```
$ atlog PLCNAME[/SLOT[/ADDR]] [опции] [REGEX ...]
```

Аргумент *PLCNAME/SLOT* задает имя ПЛК и номер слота модуля, к журналу которого происходит обращение. Имя ПЛК должно присутствовать в файле *conf.yml*, описание слота может отсутствовать. Если опциональная часть *[/SLOT]* не указана, обращение происходит к головному (master) модулю указанного ПЛК. Опциональная часть */ADDR* применяется для обращения к устройству, подключенному к внешней шине Modbus модуля AT241MG.

Опциональный аргумент *REGEX* позволяет уменьшить объем передаваемой информации, включая в выборку только записи журнала, удовлетворяющие заданному образцу – регулярному выражению *REGEX*. Описание синтаксиса регулярных выражений *POSIX* можно найти в документации ОС *GNU/Linux: man 7 regex* или [10]. Если задано несколько аргументов *REGEX*, результаты сопоставления объединяются логическим И.

3.5.1 Опциональные аргументы программы *atlog*

- *--help*: программа *atlog* выдает краткую справку по использованию и завершается

```
$ atlog –help
Usage: atlog PLCNAME [[-eirsv] REGEX ...] [-N] [-f]
       REGEX is a POSIX regular expression (man 7 regex)
       N is the number of most recent records to fetch
       -e use POSIX Extended Regular Expression syntax
       -f `follow' mode, like tail -f
       -i ignore case
       -r output in reverse chronological order
       -s let '.' match newline (\n)
       -v invert match result, like grep -v
```

- *-f /path/to/conf.yml*: задает альтернативный путь к файлу конфигурации. По умолчанию программа *atlog* использует файл конфигурации *%BASEDIR%/conf/conf.yml*, который находится исходя из пути запуска програмы *atlog (%BASEDIR%/bin/atconf)*;
- *-e*: включает расширенный синтаксис регулярных выражений [10];
- *-f*: включает «режим сопровождения»: после вывода всех заданных другими аргументами записей программа *atlog* не завершается, а продолжает работать, выводя новые записи по мере их добавления в журнал;
- *-i*: делает образец *REGEX* нечувствительным к регистру букв (*case-insensitive*);
- *-r*: выводит записи в обратном хронологическом порядке, начиная с самой недавней;
- *-s*: разрешает метасимволу '.' в регулярных выражениях сопоставляться с символом перевода строки (*LF, '\n'*) в многострочных записях журнала;

➢ *-v*: инверсия результата сопоставления записи журнала с образцом: строка, удовлетворяющая регулярному выражению, будет не включена, а наоборот пропущена при выводе.

3.5.2 Примеры обращения

Считывание и сохранение в файле *AI_module.log* всех записей журнала модуля аналогового ввода AT241AI4, находящегося в позиции 3 в сборке УСО/ПЛК AT241, присутствующего в конфигурационном файле *conf.yml* с именем *plc1*:

```
$ atlog plc1/3 > AI_module.log
```

Считывание трех последних записей журнала головного модуля ПЛК *plc103*, после чего ожидание новых записей и вывод их по мере поступления:

```
$ atlog plc103 -3f
```

Считывание всех записей журнала модуля аналогового ввода AT241AI4, кроме относящихся к каналам аналогового ввода номер 0 и номер 1 или записанных функцией app_main:

```
$ atlog plc1/3 -v 'chan[01]' -v app_main
```

3.6 Обращение к программе *atupload*

Программа *atupload* выполняет компиляцию и загрузку в ПЛК программ на языке ST IEC 61131-3. Загрузка программы не требует перезагрузки ПЛК и может выполняться на фоне с выполнения в ПЛК ранее загруженных программ, подобно запуску команд в командной строке операционной системы.

Для компиляции программа ПЛК должна быть описана в файле *conf.yml* в разделе *plcN: plcprogN*, где, в частности, указывается имя файла, содержащего текст программы на языке ST. Для запуска программы на исполнение она должна быть загружена программой *atupload*, и в конфигурации должно быть разрешено ее исполнение (*plcN: plcprogN: enable: y*). Язык конфигурации подробно описан в разделе 5.

Программа *atupload* требует два обязательных аргумента: имя ПЛК и индекса программы *plcprogN*. Допускается краткая запись индекса в виде N.

При таком запуске программы *atupload* происходит компиляция программы *plcprog1*, и при отсутствии ошибок компиляции выполняется загрузка программы в ПЛК. После успешной загрузки происходит запуск программы на исполнение, если в конфигурации присутствует параметр *plcN: plcprogN: enable: y*,

3.6.1 Опциональные аргументы программы atupload

➢ *--help*: команда *atupload* выдает краткую справку по использованию и завершается:

```
$ atupload --help
Usage: atupload PLCNAME plcprogN
       atupload -c PLCNAME plcprogN
       atupload -c FILENAME.st
       atupload [-f conffile.yml] ...
```

- *-f /path/to/conf.yml* задает альтернативный путь к файлу конфигурации. По умолчанию программа *atupload* использует файл конфигурации *%BASEDIR%/conf/ conf.yml*, который находится исходя из пути запуска программы *atupload (%BASEDIR%/bin/ atconf)*;
- *-c*: опция запрещает загрузку программы в ПЛК; выполняется только компиляция программы с выводом размера секций программы и сообщений об ошибках, если в тексте программы на языке ST есть ошибки. При вызове *atupload* с ключом *-c* допускается передача в качестве аргумента имени файла с программой вместо имени ПЛК и индекса программы, например: *upload -c test.st.*

3.6.2 Замечания по реализации

Программа *atupload* в ходе работы несколько раз вызывает системную утилиту *make* в каталоге *%BASEDIR%/st*, задавая ей разные цели компиляции и параметры. Процесс компиляции определен в файле *%BASEDIR%/st/Makefile*; этот файл не предполагает изменения пользователем, но может быть полезен для ознакомления с деталями процесса компиляции.

Компиляция и загрузка программы происходит в пять этапов:

Трансляция с языка ST в язык Си транслятором *matiec* [3] (исполняемый файл транслятора *iec2c*). Результат трансляции представлен в виде ряда файлов в каталоге *%BASEDIR%/st/build*:

```
-rw-r--r-- 1 npc21live npc21live 617 Sep 24 18:41 Config0.c
-rw-r--r-- 1 npc21live npc21live 1 Sep 24 18:41 Config0.h
-rw-r--r-- 1 npc21live npc21live 108 Sep 24 18:41 LOCATED_VARIABLES.h
-rw-r--r-- 1 npc21live npc21live 22 Sep 24 18:41 POUS.c
-rw-r--r-- 1 npc21live npc21live 122 Sep 24 18:41 POUS.h
-rw-r--r-- 1 npc21live npc21live 1344 Sep 24 18:41 program0.c
-rw-r--r-- 1 npc21live npc21live 477 Sep 24 18:41 program0.h
-rw-r--r-- 1 npc21live npc21live 677 Sep 24 18:41 Res0.c
-rw-r--r-- 1 npc21live npc21live 2253 Sep 24 18:41 VARIABLES.csv
```

Предварительная компиляция из языка Си в машинный код с использованием программ из комплекта GNU Toolchain, входящих в поставку SDK esp-idf. Главная из этих программ – кросс-компилятор языка Си для архитектуры Xtensa (*xtensa-esp32s3-elf-gcc*). В результате компиляции создаются следующие файлы:

- *build/text.bin* с образом сегмента команд;
- *build/data.bin* с образом сегмента инициализированных данных;
- *build/map.txt* с картой памяти скомпилированной программы.

Файлы *text.bin* и *data.bin* являются абсолютными загружаемыми образами: они не содержат информации о перемещении, а настроены на загрузку по заранее определенным адресам сегментов *text, data* и *bss*. «Предварительность» данного этапа компиляции состоит в том, что конкретные адреса загрузки пока не известны, они задаются произвольно. Цель предварительной компиляции – определить размеры сегментов *text, data* и *bss*.

Если программа *atupload* запущена с ключом *-c* («только компиляция»), на этом этапе она завершается.

Определение адресов загрузки. Программа atupload обращается к ПЛК по протоколу Modbus TCP с использованием нестандартной (User defined) функции *NPC21_MODBUS*, в которой передается команда *GETPLACE* с размерами сегментов *.text, .data* и *.bss* в качестве аргументов. ПЛК выделяет память для загрузки программы с указанными размерами. Результатом команды являются адреса выделенных сегментов памяти.

Окончательная компиляция: повторяется процесс компиляции из языка Си в машинный код процессора Xtensa с генерацией абсолютных образов сегментов, настроенных на полученные в ответе на команду *GETPLACE* адреса загрузки сегментов *.text, .data* и *.bss*.

Загрузка программы. Программа *atupload* обращается к ПЛК по протоколу *NPC21_MODBUS*, в которой передается команда *PLCPROG*. Аргументом команды являются заголовок с размерами сегментов, бинарные образы сегментов *.text* и *.data* и контрольные суммы сегментов (SHA256), защищающие загружаемые образы от ошибок передачи. При отсутствии ошибок передачи бинарные образы загружаются в области памяти, выделенные по команде *GETPLACE*.

После успешной загрузки программа немедленно запускается на исполнение, если выполнены остальные необходимые условия запуска (раздел 2.1.2).

3.7 Обращение к встроенной программе модуля *AT241EA*

Программа *AT241EA* взаимодействует с другими компонентами системы автоматизации по протоколу Modbus RTU в роли ведомого (*«slave»*), отвечая на обращения ведущего (*«master»*). В обращениях передаются значения измеренных модулем *AT241EA* параметров электросети. В программе модуля каждый параметр представлен именованной переменной. Имена переменных для каждого параметра и соответствие переменных регистрам Modbus приведены в таблице 3.2.

Таблица 3.2 – Переменные и регистры модуля *AT241EA*

Имя переменной	Регистр Modbus	Тип	Описание
Ua	0	float32	Действующее значение фазного напряжения U_A
Ub	2	float32	Действующее значение фазного напряжения U_B
Uc	4	float32	Действующее значение фазного напряжения U_C
$Uavg$	6	float32	Среднее действующее значение фазного напряжения U
Uab	8	float32	Действующее значение междуфазного напряжения U_{AB}
Ubc	10	float32	Действующее значение междуфазного напряжения U_{BC}
Uca	12	float32	Действующее значение междуфазного напряжения U_{CA}
$Ullavg$	14	float32	Среднее действующее значение межфазного напряжения
Ia	16	float32	Действующее значение фазного тока I_A
Ib	18	float32	Действующее значение фазного тока I_B
Ic	20	float32	Действующее значение фазного тока I_C
$Iavg$	22	float32	Среднее действующее значение фазного тока I
Pa	24	float32	Активная мощность фазы нагрузки P_A
Pb	26	float32	Активная мощность фазы нагрузки P_B
Pc	28	float32	Активная мощность фазы нагрузки P_C
$Psum$	30	float32	Суммарная активная мощность P
Qa	32	float32	Реактивная мощность фазы нагрузки Q_A
Qb	34	float32	Реактивная мощность фазы нагрузки Q_B
Qc	36	float32	Реактивная мощность фазы нагрузки Q_C
$Qsum$	38	float32	Суммарная реактивная мощность Q
$Q1a$	40	float32	Реактивная мощность фазы A на частоте первой гармоники
$Q1b$	42	float32	Реактивная мощность фазы B на частоте первой гармоники

Продолжение таблицы 3.2

Имя переменной	Регистр Modbus	Тип	Описание
$Q1c$	44	float32	Реактивная мощность фазы C на частоте первой гармоники
$Q1sum$	46	float32	Суммарная реактивная мощность на частоте первой гармоники
Sa	48	float32	Полная мощность фазы нагрузки S_A
Sb	50	float32	Полная мощность фазы нагрузки S_B
Sc	52	float32	Полная мощность фазы нагрузки S_C
$Ssum$	54	float32	Суммарная полная мощность S
pfa	56	float32	Коэффициент мощности фазы нагрузки $\cos(\varphi A)$
pfb	58	float32	Коэффициент мощности фазы нагрузки $\cos(\varphi B)$
pfc	60	float32	Коэффициент мощности фазы нагрузки $\cos(\varphi C)$
pf	62	float32	Коэффициент мощности общий $\cos(\varphi)$
wha	64	float64	Активная энергия фазы A Wh_A
whb	72	float64	Активная энергия фазы B Wh_B
whc	80	float64	Активная энергия фазы C Wh_C
wh	88	float64	Активная энергия суммарная Wh
$whpa$	96	float64	Активная энергия фазы А в положительном направлении Wh_{P_A}
$whpb$	104	float64	Активная энергия фазы В в положительном направлении Wh_{P_B}
$whpc$	112	float64	Активная энергия фазы С в положительном направлении Wh_{P_C}
whp	120	float64	Активная энергия в положительном направлении, суммарная Wh_P
$whna$	96	float64	Активная энергия фазы А в отрицательном направлении Wh_{N_A}
$whnb$	104	float64	Активная энергия фазы В в отрицательном направлении Wh_{N_B}
$whnc$	112	float64	Активная энергия фазы С в отрицательном направлении Wh_{N_C}
whn	120	float64	Активная энергия в отрицательном направлении, суммарная Wh_N
$varha$	128	float64	Реактивная энергия фазы А в отрицательном направлении Var_{h_A}
$varhb$	136	float64	Реактивная энергия фазы В в отрицательном направлении Var_{h_B}
$varhc$	144	float64	Реактивная энергия фазы С в отрицательном направлении Var_{h_C}
$varh$	152	float64	Реактивная энергия в отрицательном направлении, суммарная Var_h
$varh1$	160	float64	Реактивная энергия в 1 квадранте
$varh2$	168	float64	Реактивная энергия в 2 квадранте
$varh3$	176	float64	Реактивная энергия в 3 квадранте
$varh4$	184	float64	Реактивная энергия в 4 квадранте

БИН 201240027825
Қазақстан Республикасы, Алматы қаласы
Республика Казахстан, город Алматы
E-mail: info@ITPEngineering.com
ТОО «IT-P Engineering»

Продолжение таблицы 3.2

Имя переменной	Регистр Modbus	Тип	Описание
$vaha$	192	float64	Полная энергия фазы A V_{ah_A}
$vahb$	200	float64	Полная энергия фазы B V_{ah_B}
$vahc$	208	float64	Полная энергия фазы C V_{ah_C}
vah	216	float64	Полная энергия суммарная V_{ah}

БИН 201240027825
Қазақстан Республикасы, Алматы қаласы
Республика Казахстан, город Алматы
E-mail: info@ITPEngineering.com

4 Входные и выходные данные

Для всех утилит ПО управления (*atconf* . . . *atupload*) входными данными является конфигурационный файл *conf.yml*. Формат конфигурационного файла описан в разделе 5.

Для программы *atupload* входными данными является текст загружаемой программы на языке ST. Язык ST детально описан в стандарте IEC 61131-3 [6] и в настоящем документе не рассматривается.

Выходными данными для всех программ, входящих в состав «ПО управления», являются сообщения, выдаваемые программами на стандартный вывод и стандартный вывод ошибок ОС Linux. Сообщения, выдаваемые программами при нормальной работе, описаны в разделе 3. Сообщения, выдаваемые программами при возникновении ошибок, описаны в разделе 7.

5 Конфигурация ПЛК Ai-Talap

5.1 Конфигурационный файл *conf.yml*

Конфигурационный файл *conf.yml* является входными данными для всех утилит ПО управления (*atconf* . . . *atupload*). Файл *conf.yml* содержит древовидную структуру конфигурационных данных в формате YAML [2] с ассоциативным массивом (ключ–значение) на верхнем уровне. Ключи ассоциативного массива на первом (верхнем) уровне дерева — это имена устройств (ПЛК или УСО) в распределенной системе автоматизации. Ключи на втором уровне дерева — это имена разделов конфигурации. Пример:

```
plc1:
  clock:{...}
  slot-1: {...}
  slot1:{...}
  slot2:{...}
plc2:
  clock:{...}
  modbus\_client: {...}
  task0:{...}
  task1:{...}
  slot-2: {...}
  slot-1: {...}
  slot1:{...}
  slot2:{...}
  ...
```

Раздел с именем вида *slotN* описывает конфигурацию модуля в слоте (разъеме расширения) номер N указанного ПЛК. Остальные разделы на уровне 2 описывают конфигурацию программных подсистем, относящуюся не к конкретному слоту, а к ПЛК в целом; они интерпретируются базовым модулем, который находится в роли головного.

5.1.1 Многофайловая конфигурация

При большом объеме конфигурационных данных можно представить их не в одном файле *conf.yml*, а в виде файлового дерева. Пример из раздела 5.1 можно представить следующим файловым деревом:

```
conf.yml/bc1.yml
conf.yml/bc1/slot-1.yml
conf.yml/bc1/slot1.yml
conf.yml/bc1/slot2.yml
conf.yml/plc202.yml
conf.yml/plc202/slot-2.yml
conf.yml/plc202/slot-1.yml
conf.yml/plc202/slot1.yml
conf.yml/plc202/slot2.yml
```

Можно выбрать и другие варианты разбиения дерева конфигурации на файлы, зная правила чтения дерева конфигурации:

- разбор начинается с имени *conf.yml*, будь то файл или каталог;

➢ после чтения файла *X.yml*, дополнительно прочитывается каталог с именем X, если он существует;

➢ при чтении каталога прочитываются все записи из него с суффиксом *.yml* (как файлы, так и каталоги). Прочитанное из файла/каталога *Y.yml* поддерево конфигурационных данных добавляется в вышестоящее дерево как элемент в ассоциативном массиве с ключом Y.

5.2 Разделы конфигурации

5.2.1 Раздел *clock*

Конфигурация клиента протокола синхронизации времени SNTP. Клиент SNTP работает на головном модуле ПЛК, остальные модули получают сигналы точного времени от головного модуля по шине ПЛК.

```
clock:
  sntp_server: sntp.example.com
  sntp_server2: sntp.example2.com
  sntp_smooth: y # y=use adjtime, n=use settimeofday
```

5.2.2 Раздел *forte*

Раздел конфигурации подсистемы 4diac FORTE. Устанавливает разрешение выполнения подсистемы и привязывает переменные брокера к функциональным блокам *PUBLISH* и *SUBSCRIBE* в проекте 4diac (Для подробной информации необходимо изучить раздел 8).

Пример:

```
forte:
  enable: y
  var: [from_forte1, from_forte2..3, cmd, do1]
  var_out: [to_forte1,to_forte2..3, cmd]
```

5.2.3 Раздел *modbus_client*

Конфигурация экземпляра клиента протокола Modbus. Возможно присутствие более одного раздела *modbus_clientN*, явного ограничения на количество разделов *modbus_clientN* нет. Функция Modbus выбирается автоматически в зависимости от типа регистра (*coil, inbit, holding, inreg*).

Пример: клиент Modbus TCP (задан параметр *server*).

```
modbus_client:  # эквивалентно modbus_client0
  server: 192.168.1.60:1502  # адрес и порт сервера
  #  запись  40  регистров  fn=0x10  Write  Multiple  Registers
  holding0..39: {var_out: VALVE1..40, period: 1s}
modbus_client2:  #  N=2,  должно  отличаться  от  других  modbus_clientN
  server: 192.168.1.610  # адрес сервера. Порт 502 по умолчанию
  #  чтение  одного  регистра  300h,  fn=0x04  Read  Input  Registers
  inreg0x300: {var: sensor5, period: 1.5s}
```

Пример: клиент Modbus RTU (задан параметр *addr*). Конфигурация клиента размещена в разделе *slotN* модуля, к порту RS-485 которого подключен нужный сегмент Modbus.

```
slot-1:
  modbus_client100:
    addr: 2 #адрессервера RTU, 1..247
    holding4..23:{var:mctest204..223,period:10ms}
    holding204..205: {var_out: mctest204..205,period:2s}
```

5.2.4 Раздел *modbus_server*

Директива задает конфигурация сервера Modbus TCP IPv4 и Modbus RTU. Сервер Modbus TCP IPv6 не требует конфигурации, запускается всегда на порту 502 и используется для конфигурации.

Пример: сервер Modbus TCP (задан параметр *tcp_port*)

```
modbus_server:
  tcp_port: 502 # optional; if present, enables
  Modbus TCP
  #регистр0001        отвечаетнafn=0x3        Read
  HoldingRegisters,
  #fn=0x06Write    Single    Register,    fn=0x10
  WriteMultipleregisters
  holding1: {var_out: cmd,var:cmd}
 #регистры 0004,0005,0006,0007 отвечают на fn=0x3ReadHoldingRegisters
 holding4..7: {var_out:ai0..3}
 fussy_read:n # неопределенный регистр читаетсякак0
 fussy_write: y # запись неопределенного регистра дает ошибку Modbus
```

5.2.5 Раздел *slot*

Раздел *slotN* описывает конфигурацию модуля в слоте расширения номер N ПЛК AT241, или в позиции N в сборке УСО AT241.

Пример: конфигурация модуля дискретных выходов

```
slot1:
  module: AT241DO16
  out0..15:
  var_out: do0..15
```

Пример: конфигурация дополнительного базового модуля. Включает конфигурацию клиента Modbus RTU, подключенного к порту RS-485 данного модуля.

```
slot-2:
  module: AT241CPU
  macaddr: f4:12:fa:d5:51:bb
  rs485:
    speed: 9600 parity: N bits: 8
    stop_bits: 1
     termination: 120 # optional; 120 or none
  modbus_client1:
```

```
    test_pattern: UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
    addr: 1
```

5.2.6 Раздел *var*

В разделе *var* можно задавать начальные значения и атрибуты глобальных переменных ПЛК, для которых это требуется в конкретном приложении. Не все переменные необходимо перечислять в разделе *var*, но если переменная упоминается как *VAR_EXTERNAL* в программах ПЛК, она должна присутствовать и иметь атрибут *init*.

Пример:

```
var:
  # 01-oscillator-stop
  start: {retain: y, init: 0}
  out1: 1 # упрощенный синтаксис; эквивалентно {init: 1}
  stop0: 0
  # snake exec: 1
  mw1: 0
  inc1: 0
  stop1: 0
  cmd: {multisource: y, init: 0}
```

Атрибут *retain: y* устанавливает режим сохранения данной переменной в энергонезависимой памяти. Последнее записанное значение будет восстановлено после перезагрузки.

Атрибут *multisource: y* отменяет для данной переменной общее правило «у переменной только один издатель» (раздел 2.1.1). При единственном издателе системное ПО гарантирует, что после обновления переменной все подсисемы ПЛК получат ее новое значение (почти одновременно, с небольшой вариацией задержки). После включения режима multisource этой гарантии больше нет: если два издателя публикуют новые значения в быстрой последовательности, возможны нежелательные эффекты, вплоть до разных значений переменной в разных подсистемах. Корректность работы в режиме multisource гарантируется, если промежуток времени между обновлениями переменной больше, чем время распространения по всем подсистемам ПЛК (которое зависит от аппаратной конфигурации конкретного ПЛК). Ответственность за соблюдение этого условия возлагается на прикладного программиста.

5.2.7 Раздел *wdt*

Настройка параметров сторожевых таймеров («*watchdog timers*»). В каждом модуле ПЛК Ai-Talap работает сторожевой таймер, способный выполнить аппаратный сброс модуля для автоматического восстановления работоспособности модуля после сбоя. При нормальной работе головной модуль ПЛК регулярно рассылает сообщения с указанием периода t, при получении которых сторожевой таймер откладывает сброс на t секунд. Если какой-то модуль из-за сбоя не получает этих сообщений или не может их обработать («подвис», перегружен и т.д.), сторожевой таймер сбрасывает модуль, модуль перезагружается и возвращается в работу.

Пример:

```
wdt:
  bus: 10s # период сторожевого таймера в подчиненных модулях ПЛК
```

```
  self: 30s # период сторожевого таймера в головном
```

При отсутствии в конфигурации раздела *wdt* сторожевой таймер в базовых модулях настраивается на период t = 3600с, в модулях ввода/вывода на период t = 300с.

5.3 Параметры конфигурации

В настоящем разделе в алфавитном порядке описываются параметры (атрибуты) конфигурации, которые могут встречаться в разных разделах на разных уровнях дерева конфигурации. Как правило, параметр с одним именем имеет один и тот же смысл независимо от раздела, где он встречается, поэтому его достаточно описать один раз. В редких случаях, когда параметр в разных разделах интерпретируется по-разному, он описывается несколько раз с указанием контекста.

5.3.1 Параметр *direct (slot:iec104_server:io)*

Параметр *direct: y* – объект io протокола IEC-104 воспринимает прямые (непосредственные) команды. Стоит по умолчанию.

Параметр *direct: n* – объект io протокола IEC-104 воспринимает только команды с предварительным выбором. На прямые (непосредственные) команды отвечает отрицательным подтверждением ACTCON.

5.3.2 Параметр *interval*

Задает минимальный временной интервал между действиями для объекта, в котором находится параметр.

Пример: ограничение максимально частоты отправки значений модулем аналогового ввода при быстрых изменениях напряжения на входе:

```
module: AT241AI4
out0:
  mode:0-10V6/
  resolution:0.0001 # игнорировать изменения менее 0.1мВ
  interval: 0.1s # максимальная частота отправки 10Гц
  period: 0.5s # минимальная частота отправки 2Гц
  var: sensor1
```

5.3.3 Параметр *module (slot)*

Обязательный параметр каждого раздела *slotN*, указывающий тип (артикул) модуля, установленного в слот *N* (или находящегося в позиции *N* в сборке УСО AT241). Каждым модуль при загрузке конфигурации проверяет значение параметра на соответствие своему типу; в случае несовпадения конфигурация не принимается.

Пример: *module AT241AI4*

Важно!!!

В разных версиях прошивок модулей расширения, а также самого ПЛК могут происходить ошибки при написании имен AT241…, поэтому при возникновении ошибок рекомендуется использовать следующие наименование модулей:

```
plc1:
  …
  slot-1:
    module: bcbase #AT241CPU
```

```
  …
slot1:
  module: bcdi #AT241DI16
  in0..15:
    var: di0..15

slot2:
  module: bcdo #AT241DO16, представлен синтаксис в формате массива
  out0..15:
    var_out: do0..15
    init: last

slot3:
  module: bcai #AT241AI4
  in0..3:
    mode: 0-10V
    range: 2  # in "natural" units = 2mV
    interval: 0.5s
    period: 1s
    natural: [0, 10000] # report voltagi in mV
    var: ai0..3

slot4:
  module: bcao #AT241AO4, представлен построчный синтаксис
  out0: {mode: 0-10V, var_out: ao0, natural: [0, 10000]}
  out1: {mode: 0-10V, var_out: ao1, natural: [0, 10000]}
  out2: {mode: 0-10V, var_out: ao2, natural: [0, 10000]}
  out3: {mode: 0-10V, var_out: ao3, natural: [0, 10000]}
```

5.3.4 Параметр *macaddr (slot)*

Обязательный параметр для разделов *slot*, в которые установлены модули с интерфейсом Ethernet; используется для формирования адреса *IPv6 link local* при обращениях к модулю с управляющего компьютера.

Пример:

```
plc1:
  …
  slot-1:
    module: bcbase #AT241CPU
    macaddr: f4:12:fa:dd:c6:17
    …
  slot1:
    module: bcdi #AT241DI16
    in0..15:
```

```
var: di0..15
```

5.3.5 Параметр *quality (slot:iec104_server:command, meas)*

Параметр *quality: n* – стоит по умолчанию.

Параметр *quality: y* – включает для конкретного объекта IEC-104 режим добавления описателя качества [12] при передаче данного объекта по протоколу МЭК-104. Биты описания качества происходят от источника значения переменной: например, программы ПЛК или объекта *ioN*, являющегося источником данной переменной. Если источник значения переменной (например, *modbus_server*) не поставляет битов качества, передается нулевой описатель, означающий наилучшее качество значения.

Если задан *slot:iec104_server,* то этот параметр задает значение параметра *quality* по умолчанию для всех объектов *io* данного сервера протокола IEC-104.

5.3.6 Параметр *select (slot:iec104_server:io)*

Параметр *select* управляет обработкой входящих команд в режиме «выбора и исполнения» [11]. Входящее сообщение с установленным битом S/E [12] воспринимается как директива предварительного выбора (SELECT).

Параметр *select: auto* – модуль немедленно отвечает на директиву выбора положительным подтверждением (*ACTCON*) и начинает отсчет времени (*select_timeout*), в течение которого должна прийти выбранная команда (*EXCO*). Если до поступления команды поступит новая директива выбора для того же объекта *io*, модуль ответит отрицательным подтверждением *ACTCON*. После поступления команды передается подтверждение (положительное, если данные в команде совпадают с данными предварительного выбора), объект *io* выходит из состояния ожидания команды и при поступлении новой директивы выбора снова ответит положительным подтверждением *ACTCON*. Данный параметр назначен по умолчанию.

Параметр *select: ind* – модуль сигнализирует о возникшем событии выбора программам ПЛК, работающим на базовом модуле, путем передачи предлагаемого нового значения объекта *io* в атрибуте *select_ind* переменной объекта (*io:var:name*). Программа ПЛК подтверждает готовность к исполнению команды (*EXCO*) установкой атрибута *select_res* равным 1, или сообщает о неготовности установкой атрибута *select_res* равным 0. Механизм подтверждения работает корректно, даже если новое значение *select_ind* совпадает с значением, установленным предыдущей командой, поскольку программа работает по событиям изменения атрибутов, которые генерируются даже если новое значение атрибута совпадает с предыдущим.

В случае получения отрицательного подтверждения *select_res = 0* модуль отвечает на директиву выбора отрицательным подтверждением *ACTCON* и объект *io* становится готов к приему новой директивы выбора.

В случае получения подтверждения *select_res = 1* модуль отвечает на директиву выбора положительным подтверждением (*ACTCON*) и начинает отсчет времени *select_timeout*, в течение которого должна прийти выбранная команда (*EXCO*). Если до поступления команды поступит новая директива выбора для того же объекта *io*, модуль ответит отрицательным подтверждением *ACTCON*. После поступления команды объект *io* выходит из состояния ожидания команды и становится готов к приему новой директивы выбора.

Для задания время ожидания команды используется параметр *select_timeout,* например:

```
select_timeout: 5s
```

При использовании в контексте *slot:iec104_server* параметр влияет на все объекты *io* данного сервера, не имеющие собственной установки *select_timeout*.

5.3.7 Параметр *period*

В контексте выполнения программного кода (*task*) задает временной интервал исполнения программы ПЛК. По смыслу соответствует параметру *CONFIGURATION: RESOURCE: TASK: INTERVAL стандарта IEC 61131* (заменяет его). Пример:

```
task0:
  period: 10ms
```

В других контекстах задает временной интервал между регулярными действиями для объекта, в котором находится параметр. Специальное значение регулярные действия для объекта.

Пример: период опроса сервера клиентом Modbus (смотреть раздел 5.2.3):

```
modbus_client5:
  period: 1s
```

Пример: период отправки значения модулем аналогового ввода при постоянном напряжении на входе:

```
plc1:
  …
  Slot1:
    module: bcai #AT241AI4
    in0..3:
      mode: 0-10V
      interval: 0.5s #максимальная частота отправки 2 Гц
      period: 1s #минимальная частота отправки 1 Гц
      var: ai0..3
```

5.4 Примеры конфигурации

Первый пример посвящен конфигурации модуля IEC104:

```
slot7:
  module: AT340IEC104

  # MAC-адрес данного модуля, обязательный параметр
  macaddr: f4:12:fa:d4:d5:7f

  # опциональная конфигурация статических параметров TCP/IP
  # если задана, отключает DHCP
  #ipaddr: 192.168.1.90/24
  #gateway: 192.168.1.60
```

```
# конфигурация первого сервера МЭК-104
iec104_server1:
  # TCP-порт сервера, по умолчанию 2404
  #tcp_port: 2404

  # ограничение на число одновременно подключенных клиентов
  # по умолчанию 50
  #max_connections: 50

  # опциональный список дозволенных клиентов
  # по умолчанию принимаются все соединения
  #allow: [192.168.1.60, 192.168.1.61]

  # параметры уровня соединения МЭК-104 (APCIP)
  conn_params:
    k: 12
    w: 8
    t0: 10s
    t1: 15s
    t2: 10s
    t3: 20s

  # параметры прикладного уровня
  #app_params:
  #  type_id_size: 1
  #  vsq_size: 1
  #  cot_size: 2
  #  originator_address: 0
  #  ca_size: 1
  #  ioa_size: 3
  #  asdu_maxsize: 249

  # Прием C_CS_NA_1(103) | Clock synchronization command
  clock_sync:
    #period: 0
    timeout: 1h

  # описание информационных объектов
  meas5000:
    var_out: myvar1

  # все возможные типы
  type: float
```

БИН 201240027825
Қазақстан Республикасы, Алматы қаласы
Республика Казахстан, город Алматы
E-mail: info@ITPEngineering.com


```
    # возможные форматы времени
    time: 56

    meas70:
      var_out: myvar20
      type: float
      range: 0.05
      interval: 1s

    meas71:
      var_out: myvar30
      type: bitstring
      interval: 3m

    io10000:
      var: myvar40
      type: singlepoint
      interval: 0.1s

    io16:
      var: myvar41
      type: bitstring

    interrogation:
      period: 0
      list: [meas5000, meas70]

    interrogation3:
      list: [meas70, meas71]

    queue:
      size: 1000
      mode: single

  # можно конфигурировать несколько серверов с разными CA
  #iec104_server2:

iec104_client0:
  server: 192.168.1.205:2404
  originator_address: 3

  conn_params:
    k: 12
    w: 8
```

БИН 201240027825
Қазақстан Республикасы, Алматы қаласы
Республика Казахстан, город Алматы
E-mail: info@ITPEngineering.com


```
  t0: 10s
  t1: 15s
  t2: 10s
  t3: 20s

clock_sync:
  period: 10m

read81:
  var: xyz1
  period: 30s

read1281:
  var: abcd
  period: 60s

interrogation:
  period: 10m

interrogation3:
  period: 1m

test:
  period: 30s
  tsc: 0x4938

command5000:
  var_out: myvar55
  type: float
  period: 5s

command5001:
  var_out: myvar56
  type: bitstring
  period: 50s

command5010:
  var_out: myvar60
  type: stepposition
  interval: 0.5s

command5011:
  var_out: myvar61
  type: singlepoint
```

```
    interval: 0.1s

  io81:
    var: myvar50
    type: float
    range: 0.1
    interval: 10ms

  io200..210:
    var: myvar55..65
    type: bitstring
    interval: 1s

  io220+5:
    var: myvar120+5
    type: singlepoint
    interval: 1s

  io0x1004:
    var: myvar115..119
    type: doublepoint
    interval: 1s
  #количество клиентов в конфигурации неограничено явно, только
  #объемомпамяти.
  #iec104_client2:
  #...
```

Пример полной конфигурации ПЛК в сборке: ПЛК, Модули DI, DO, AI, AO:

```
plc1:
  clock: # Настройка синхронизации времени
    TZ: KZT # Настройка временной зоны
    sntp_server: pool.ntp.org # Задание сервера точного времени
    sntp_smooth: y  # y - включение синхронизации, n - использование заданного
  времени

  ipaddr: 192.168.2.1/24 # Задание IP-адреса ПЛК

  var: # Объявление переменных и их начальных значений
    temp: {init: 0}
    nom_val: {init: 20}
    hi_sensor: {retain: y, init: 500}
    lo_sensor: {retain: y, init: 0}
    temp_per: {retain: y, init: 0}
```

```
forte:
  enable: y #Активация режима Forte для передачи значений переменных в блок 4diac Forte
  var: [do0, temp, temp_per] # Задание переменных, которые будут получены из Forte (блок Publish)
  var_out: [di5, ai0, nom_val, hi_sensor, lo_sensor] # Задание переменных, которые будут отправлены в Forte (блок Subscribe)

slot-1:
  module: bcbase # AT241CPU
  macaddr: f4:12:fa:dd:c6:17

  modbus_server: # Конфигурация Modbus-Master
    tcp_port: 502
    holding0..3: {var_out: ai0..3} # Пример того, что можно задавать массивом
    holding4: {var_out: temp} # Выполнение функции 03 для отправки значения переменной temp с номером регистра 4

slot1:
  module: bcdi # AT241DI16
  in0..15: # Инициализация входных переменных модуля
    var: di0..15 #И их объявление

slot2:
  module: bcdo #AT241DO16
  out0..15: # Инициализация выходных переменных модуля
    var_out: do0..15 #И их объявление

slot3:
  module: bcai #AT241AI4
  in0..3: # Инициализация входных переменных модуля
    mode: 0-10V # Настройка режима канала на напряжение от 0 до 10 В
    range: 2  # Настройка зоны нечувствительности в mV
    interval: 0.5s # Интервал считывания, также максимальное время обновления - 2 Гц
    period: 1s # Период считывания, также минимальное время обновления - 1 Гц
    natural: [0, 10000] # Масштабирование сигнала, можно делать напрямую, но для унификации рекомендуется использовать одни и те же цифры для всех каналов
    var: ai0..3 # Объявление переменных модуля

slot4:
  module: bcao #AT241AO4
  out0..3: # Инициализация выходных переменных модуля
    mode: 0-10V # Настройка режима канала на напряжение от 0 до 10 В
```

```
    natural: [0, 10000] # Масштабирование сигнала, можно делать напрямую, но
для унификации рекомендуется использовать одни и те же цифры для всех каналов
    var_out: ao0..3 # Объявление переменных модуля
```

6 Диагностика

Встроенное ПО каждого модуля ПЛК Ai-Talap формирует и регулярно передает на головной модуль слова состояния, содержащие биты состояния и ошибок. На головном модуле полученные от подчиненных модулей слова состояния доступны в переменных брокера *sys.alarmN* и *sys.statusN*, где N – позиционный номер модуля. Для модулей с отрицательными позиционными номерами используются переменные *sys.alarm_N* и *sys.status_N*. Диагностическая информация головного модуля ПЛК дополнительно доступна в переменнных *sys.alarm* и *sys.status*, независимо позиционного номера модуля, исполняющего в данный момент роль головного.

Переменные *sys.alarmN* содержат биты ошибок. Назначение битов в переменных *sys.alarmN* одинаковое для всех модулей ПЛК Ai-Talap. Головной модуль дополнительно формирует переменную *sys.allalarm*, содержащую побитовое ИЛИ текущего значения переменных *sys.alarmN* всех модулей; при нормальной работе ПЛК все переменные *sys.alarmN* и переменная *sys.allalarm* имеют значение «0».

Переменные *sys.statusN* содержат биты состояния типов E и I. Биты типа E конкретизируют причины установки битов ошибок *sys.alarmN*; эти биты имеют значение «0» при нормальной работе ПЛК. Биты типа I носят информационный характер и могут иметь значения «0» или «1» при нормальной работе ПЛК. Назначение битов *sys.statusN* специфично для каждого типа модуля.

Переменные состояния доступны для считывания по протоколу npc21_modbus, а также могут передаваться в прикладные программы ПЛК и другие подсистемы средствами конфигурации. Например, можно настроить доступ к переменным состояния по протоколу Modbus:

```
modbus_server:
  inreg101..120: {var_out: sys.status1..20}
  inreg201..220: {var_out: sys.alarm1..20}
  inreg299: {var_out: sys.allalarm}
```

Назначение битов переменных состояния для каждого типа модулей ПЛК Ai-Talap описано в таблицах 6.1–6.5.

Таблица 6.1 – Назначение битов *sys.alarmN* всех модулей

Номер бита	Обозн.	Тип	Описание
15	MA	E	«*Module absent*»: модуль не присылал диагностическую информацию головному модулю в течение последних 8 с (отсутствует или неисправен)
14	UE	E	«*Unclassified error*»: любая ошибка, не имеющая отдельного бита в sys.alarm. Детализация в sys.statusN и/или liblog резерв
13		E	Резерв
12	MC	E	«*Modbus client*»: ошибка клиента Modbus: нет ответа сервера, ошибочный ответ сервера (exception), ошибка CRC
11	VB	E	«*Battery voltage*»: неисправность автономного источника питания часов реального времени

Продолжение таблицы 6.1

Номер бита	Обозн.	Тип	Описание
10	RU	E	«*PLC run mode*»: выполнение прикладных программ ПЛК остановлено (переменная sys.run имеет значение «0», установленное по ошибке выполнения прикладной программы или по команде оператора)
9	IP	E	«*IP address*»: модуль не имеет IP-адреса (ни статического ни назначенного по протоколу DHCP)
8	PA	E	«*Power absent*»: модуль является инжектором питания шины ПЛК (AT241BC, AT241CPU, AT340EXT, AT340PWR), и в последние 3с напряжение на разъеме внешнего питания модуля не приходило в норму (24В ±20%)
7	PW	E	«*Power error*»: модуль является инжектором питания шины ПЛК, и за последние 3с были замечены отклонения напряжения на разъеме внешнего питания модуля от нормы (24В ±20%)
6	PI	E	«*I/O power*»: в последние 3с замечены отсутствие или отключения от нормы дополнительного питания модуля ввода-вывода (на модулях AT241DO16, AT340DO32, AT241MG)
5	AC	E	«*Alternate current*»: замечено присутствие переменного напряжения 220В 50Гц на порту ввода-вывода или входе дополнительного напряжения питания модуля ввода-вывода (на модулях AT241DO16, AT340DO32, AT241DI16, AT340DI32). Порт переведен в состояние защиты
4	VI	E	«*Internal voltage*»: замечено отклонение от нормы одного из внутренних напряжений модуля (самодиагностика)
3	CH	E	«*I/O channel*»: один или более канал ввода-вывода находится в состоянии ошибки; детализация в sys.statusN и/или liblog
2	NC	E	«*Need configuration*»: в модуле не загружена конфигурация
1	BF	E	«*Bus failure*»: долговременная ошибка шины ПЛК, передача данных невозможна
0	BE	E	«*Bus error*»: за последние 3 с были замечены сбои в работе шины ПЛК, например ошибки CRC

Таблица 6.2 – Назначение битов *sys.statusN* модуля AT241BC и AT241CPU

Номер бита	Обозн.	Тип	Описание
15	BM	I	Модуль работает в роли головного модуля шины
14		I	Резерв
13		I	Резерв
12		I	Резерв
11		I	Резерв
10		I	Резерв
9		I	Резерв
8		I	Резерв
7		I	Резерв
6		I	Резерв
5		I	Резерв

Продолжение таблицы 6.2

Номер бита	Обозн.	Тип	Описание
4	S0	I	Порт COM работает в режиме ведомого (сервера)
3		I	Резерв
2		I	Резерв
1		I	Резерв
0	M0	I	Порт COM работает в режиме ведущего (клиента)

Таблица 6.3 – Назначение битов *sys.statusN* модуля AT241DI16 и AT241DO16

Номер бита	Обозн.	Тип	Описание
15	C15	E	Ошибка на канале 15
14	C14	E	Ошибка на канале 14
13	C13	E	Ошибка на канале 13
12	C12	E	Ошибка на канале 12
11	C11	E	Ошибка на канале 11
10	C10	E	Ошибка на канале 10
9	C9	E	Ошибка на канале 9
8	C8	E	Ошибка на канале 8
7	C7	E	Ошибка на канале 7
6	C6	E	Ошибка на канале 6
5	C5	E	Ошибка на канале 5
4	C4	E	Ошибка на канале 4
3	C3	E	Ошибка на канале 3
2	C2	E	Ошибка на канале 2
1	C1	E	Ошибка на канале 1
0	C0	E	Ошибка на канале 0

Таблица 6.4 – Назначение битов *sys.statusN* модуля AT241AI4

Номер бита	Обозн.	Тип	Описание
24	TR	E	Ошибки датчиков температуры модуля (Tref)
23			Резерв
22			Резерв
21			Резерв
20	HP	E	Перегрузка преобразователя питания каналов
19	F3	E	Неклассифицированная ошибка на канале 3
18	F2	E	Неклассифицированная ошибка на канале 2
17	F1	E	Неклассифицированная ошибка на канале 1
16	F0	E	Неклассифицированная ошибка на канале 0
15	B3	E	Обрыв на канале 3
14	B2	E	Обрыв на канале 2
13	B1	E	Обрыв на канале 1
12	B0	E	Обрыв на канале 0
11	S3	E	КЗ на канале 3
10	S2	E	КЗ на канале 2
9	S1	E	КЗ на канале 1

Продолжение таблицы 6.4

Номер бита	Обозн.	Тип	Описание
8	S0	E	КЗ на канале 0
7	P3	E	Сработала защита на канале 3
6	P2	E	Сработала защита на канале 2
5	P1	E	Сработала защита на канале 1
4	P0	E	Сработала защита на канале 0
3	D3	I	Канал 3 не описан в конфигурации
2	D2	I	Канал 2 не описан в конфигурации
1	D1	I	Канал 1 не описан в конфигурации
0	D0	I	Канал 0 не описан в конфигурации

Таблица 6.5 – Назначение битов *sys.statusN* модуля AT241AO4

Номер бита	Обозн.	Тип	Описание
21	V5	E	Отклонение внутреннего напряжения 5В от нормы
20	V10	E	Отклонение внутреннего напряжения 10В от нормы
19			Резерв
18			Резерв
17			Резерв
16	HP	E	Перегрузка преобразователя питания каналов
15	B3	E	Обрыв на канале 3
14	B2	E	Обрыв на канале 2
13	B1	E	Обрыв на канале 1
12	B0	E	Обрыв на канале 0
11	S3	E	КЗ на канале 3
10	S2	E	КЗ на канале 2
9	S1	E	КЗ на канале 1
8	S0	E	КЗ на канале 0
7	F3	E	Неклассифицированная ошибка на канале 3
6	F2	E	Неклассифицированная ошибка на канале 2
5	F1	E	Неклассифицированная ошибка на канале 1
4	F0	E	Неклассифицированная ошибка на канале 0
3	D3	I	Канал 3 не описан в конфигурации
2	D2	I	Канал 2 не описан в конфигурации
1	D1	I	Канал 1 не описан в конфигурации
0	D0	I	Канал 0 не описан в конфигурации

БИН 201240027825
Қазақстан Республикасы, Алматы қаласы
Республика Казахстан, город Алматы
E-mail: info@ITPEngineering.com
ТОО «IT-P Engineering»

7 Сообщения

Программа передает ведущему модулю информацию об ошибках исполнения команд и других событиях в текстовом виде в пакетах LOG.

7.1 Сообщение «*OK conf NNNNNNNN*»

Сообщение он успешном разборе конфигурации. Подстрока NNNNNNNN представляет собой первые 8 символов хэш-суммы SHA256 от конфигурации в формате JSON. Эта подстрока предназначена для идентификации версии конфигурации в системном журнале.

7.2 Сообщение «*expected var name after var_out*»

Сообщение об ошибке разбора: после ключевого слова *var_out* должно следовать имя переменной.

7.3 Сообщение «*out of memory*»

Сообщение о нехватке оперативной памяти. Не должно возникать при нормальной работе программы. Может возникнуть при чрезмерной длине пакета конфигурации, или из-за ошибки в программе.

7.4 Сообщение «*unexpected item*»

Сообщение об ошибке разбора: обнаружено недопустимое имя параметра. Допустимые имена параметров перечислены в разделе 5.

7.5 Сообщение «*object expected*»

Сообщение об ошибке разбора: нарушение структуры конфигурации, вместо подраздела («*object*») обнаружен другой элемент, например строка.

7.6 Сообщение «*expected "module: DO"*»

Сообщение об ошибке разбора: в конфигурации отсутствует параметр *module:* или значение параметра *module* не совпадает с типом модуля («DO»).

7.7 Сообщение «*bad out-channel number, expected out0 ... out63*»

Сообщение об ошибке разбора: в параметре *outN* указан несуществующий номер выхода N.

7.8 Сообщение «*duplicate out-channel*»

Сообщение об ошибке разбора: в конфигурации обнаружено более одного подраздела *outN* с одним и тем же N.

7.9 Сообщение «*expected `module', 'init' or `outN'*»

Сообщение об ошибке разбора: обнаружено недопустимое имя параметра на верхнем уровне. Допустимые имена перечислены в разделе 5.

7.10 Сообщение «*cJSON_Parse failed*»

Сообщение об ошибке разбора: нарушение формата JSON. Не должно возникать при использовании штатных средств трансляции и загрузки конфигурации.

7.11 Сообщение «*timeout waiting reply from the slot*»

Сообщение говорит о том, что заданный в команде слот не доступен и/или неправильно указан слот. Рекомендуется проверить конфигурацию ПЛК как на физическом уровне, проверив, в какой последовательности подключены модули, так и в конфигурационном файле. Если же все указано правильно, то это указывает на ошибку модуля и его идентификации. Для этого необходимо ввести команду в командную строку (пример представлен для ПЛК, имя в конфигурации которого plc1):

```
$ for i in {55..63}; do
  atcmd plc1/$i sys.devtype
done
```

Если в результате выполнения этой команды был найден потерянный модуль, то необходимо произвести обновление (либо откат) ПО для данного модуля. Для этого нужно воспользоваться командой *atota plc1/J "Путь к файлу прошивки"*, где J – номер слота модуля, который был получен при выполнении поиска недоступного модуля. Если при попытке обновления произошла ошибка, тогда нужно собрать конфигурацию из головного модуля и модуля, который нужно перепрошить, обновить конфигурацию ПЛК, далее повторить все действия, начиная с обнаружения «пропавшего модуля» и заканчивая перепрошивкой модуля, затем проверить, работоспособен ли модуль. Если модуль неработоспособен, при этом он определяется в нужном слоте (проверка происходит посредством команды *atcmd plc1/k sys.devtype,* где k – номер правильного слота), то следует перепрошить модуль командой *atota plc1/k "Путь к файлу прошивки",* где k – номер правильного слота.

7.12 Сообщение «Configuration status: slot4: out0..3: expected `module' or `chanN'»

При появлении данной ошибки во время применения команды *atconf* следует проверить правильность знаков, символов и структуры конфигурации. Если же при проверке не было выявлено ошибок, тогда следует заменить ключ *outN* на *chanN*.

7.13 Сообщения, связанные с ошибкой Modbus

Ошибки, которые связаны с Modbus, зачастую имеют сообщения *«Modbus TCP connection failed», «got SIGPIPE, probably Modbus connection failed»*.

Если обычная перезагрузка ПЛК не помогла, то этому способствует 3 основных причины.

Первая причина заключается в том, что в сети есть устройство, которое имеет такой же IP-адрес, как и сам контроллер. В таких случаях рекомендуется, подключить ПЛК к управляющему устройству (ноутбук, компьютер) и попробовать загрузить конфигурацию. Если ошибка не была устранена, значит необходимо разобрать следующую причину.

Вторая причина – неправильно выставлен MAC-адрес устройства, что мешает корректному поднятию протокола взаимообмена. В этом случае стоит перепроверить ключ *macaddr* на правильность ввода.

Третья причина – внутренний сбой прошивки ПЛК. В этом случае необходимо перепрошить модуль CPU (BC) командой *atota*. Если же это не помогло, значит, причина вызвана более глубокими сбоями системы и требует незамедлительного обращения к интегратору, либо поставщику данного контроллера.

8 Разработка программного обеспечения в среде 4diac

8.1 Подготовка конфигурации ПЛК

Как уже было описано выше, один из возможных (и рекомендуемых) способов программирования ПЛК Ai-Talap 241 – программирование согласно стандарту МЭК 61499 в свободно распространяемой среде 4diac. Специально для поддержки этой возможности ядро операционной системы ПЛК содержит в себе среду *4diac forte* для интерпретации кода в машинный язык. Для использования этой функции в конфигурации ПЛК необходимо включить этот модуль путем установления ключа *enable: y* для раздела *forte*.

Для более ясного понимания предлагается рассмотреть программирование в этой среде на примере следующей задачи.

У некого технологического процесса есть показания температуры. Датчик выдает в качестве контрольного сигнала напряжение от 0 до 10 В. Датчик имеет диапазон от 0 до 500℃. Номинальная температура равна 20℃. Задача будет стоять такой: если запущена кнопка «Начать измерение», то происходит считывание температуры и ее масштабирование по диапазонам датчика. Если температура больше номинальной, то необходимо выдать предупредительный сигнал и держать его до тех пор, пока температура не станет равна или меньше номинальной. Также, необходимо производить расчет – в каком соотношении находится текущая температура от диапазона датчика. По Modbus нужно передать значение температуры и всех аналоговых входов. Порт Modbus оставить по умолчанию.

Конфигурация для данной задачи, которая уже упоминалась в разделе 5.4, представлена ниже.

```
plc1:
  clock: # Настройка синхронизации времени
    TZ: KZT # Настройка временной зоны
    sntp_server: pool.ntp.org # Задание сервера точного времени
    sntp_smooth: y  # y - включение синхронизации, n - использование заданного
  времени

  ipaddr: 192.168.2.1/24 # Задание IP-адреса ПЛК

  var: # Объявление переменных и их начальных значений
    temp: {init: 0}
    nom_val: {init: 20}
    hi_sensor: {retain: y, init: 500}
    lo_sensor: {retain: y, init: 0}
    temp_per: {retain: y, init: 0}

  forte:
    enable: y #Активация режима Forte для передачи значений переменных в блок
  4diac Forte
    var: [do0, temp, temp_per] # Задание переменных, которые будут получены из
  Forte (блок Publish)
    var_out: [di5, ai0, nom_val, hi_sensor, lo_sensor] # Задание переменных,
  которые будут отправлены в Forte (блок Subscribe)

  slot-1:
```

```
  module: bcbase # AT241CPU
  macaddr: f4:12:fa:dd:c6:17

  modbus_server: # Конфигурация Modbus-Master
    tcp_port: 502
    holding0..3: {var_out: ai0..3} # Пример того, что можно задавать массивом
    holding4: {var_out: temp} # Выполнение функции 03 для отправки значения
переменной temp с номером регистра 4

slot1:
  module: bcdi # AT241DI16
  in0..15: # Инициализация входных переменных модуля
    var: di0..15 #И их объявление

slot2:
  module: bcdo #AT241DO16
  out0..15: # Инициализация выходных переменных модуля
    var_out: do0..15 #И их объявление

slot3:
  module: bcai #AT241AI4
  in0..3: # Инициализация входных переменных модуля
    mode: 0-10V # Настройка режима канала на напряжение от 0 до 10 В
    range: 2  # Настройка зоны нечувствительности в mV
    interval: 0.5s # Интервал считывания, также максимальное время обновления
- 2 Гц
    period: 1s # Период считывания, также минимальное время обновления - 1
Гц
    natural: [0, 10000] # Масштабирование сигнала, можно делать напрямую, но
для унификации рекомендуется использовать одни и те же цифры для всех каналов
    var: ai0..3 # Объявление переменных модуля

slot4:
  module: bcao #AT241AO4
  out0..3: # Инициализация выходных переменных модуля
    mode: 0-10V # Настройка режима канала на напряжение от 0 до 10 В
    natural: [0, 10000] # Масштабирование сигнала, можно делать напрямую, но
для унификации рекомендуется использовать одни и те же цифры для всех каналов
    var_out: ao0..3 # Объявление переменных модуля
```

Как видно из данной конфигурации, в первую очередь указывается имя ПЛК, далее настраивается синхронизация времени – временная зона, сервер SNTP и активация синхронизации (в данном случае, так как не настроен шлюз с помощью ключа, который указывается сразу под ключом *ipaddr*; *gateway: 0.0.0.0*, где вместо 0.0.0.0 пишется настоящий адрес шлюза; это не имеет смысла, но в рамках демонстрации все же было приведено), далее

указывается IP-адрес ПЛК, после необходимо произвести объявление переменных (чтобы они были доступны с командной строки посредством *atcmd* (подробнее смотреть раздел 3.3.1)).

После этого нужно включить режим forte, установив ключ *enable* как *y.* Так как 4diac forte построен по принципу распределенных систем как издатель-подписчик, то необходимо указать, какие переменные нужно издать (передать в среду forte) и записать их в раздел *var_out;* а на какие переменные нужно подписаться (взять из среды) и записать их в раздел *var.*

Далее, конфигурируется модуль AT241CPU, в котором необходимо настроить Modbus. Для удобства использования функции прописываются так, как они называются в протоколе, в данном случае будет использована функция holding. После слова функции указывается номер регистра, начиная с 0. Потом конфигурируются все остальные модули. Для подробной информации о конфигурировании необходимо обратиться к разделу 5.

Также стоит отметить, что в обязательном порядке необходимо правильно вводить MAC-адрес в *macaddr,* иначе конфигурация не загрузится, либо могут быть помехи, так как он используется для формирования IPv6 адреса и необходим при обнаружении или загрузке конфигурации посредством ключа *–-bymac* (для подробной информации смотреть раздел 1.1).

После загрузки конфигурации посредством команды *atconf* (подробнее смотреть раздел 1.1.4) в ПЛК, можно приступать к разработке ПО в 4diac.

8.2 Создание проекта и настройка параметров

Для создания проекта в 4diac (для программирования этого ПЛК рекомендуется использовать версию 2.0.1) необходимо в меню File выбрать New, далее 4diac IDE Project, ввести название проекта и путь его сохранения (по умолчанию, все сохраняется в папке окружения, выбранной при старте программы, как показано на рисунке 8.1.

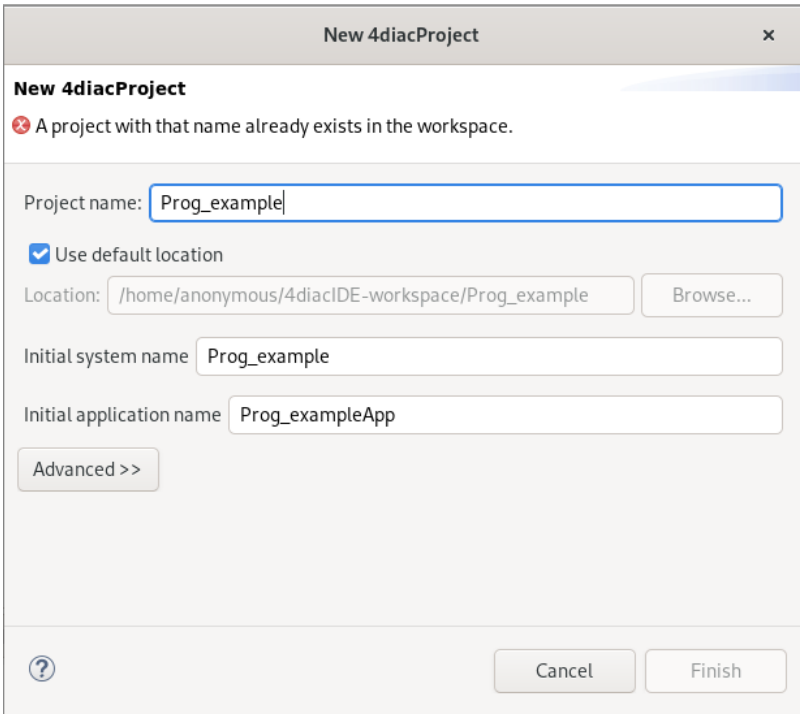

Рисунок 8.1 – Создание проекта

После создания проекта в окне *System Explorer* появится проект, как показано на рисунке 8.2. Он состоит из раздела типа *System* и раздела *Type Library.*

System – раздел, включающий в себя пространство для разработки ПО типа *App*, который автоматически генерирует название (в данном проекте – *Prog_exampleApp),* а также

системной конфигурации *System Configuration*, где производится настройка параметров устройства.

Type Library – раздел, который содержит в себе стандартный библиотеки функциональных блоков.

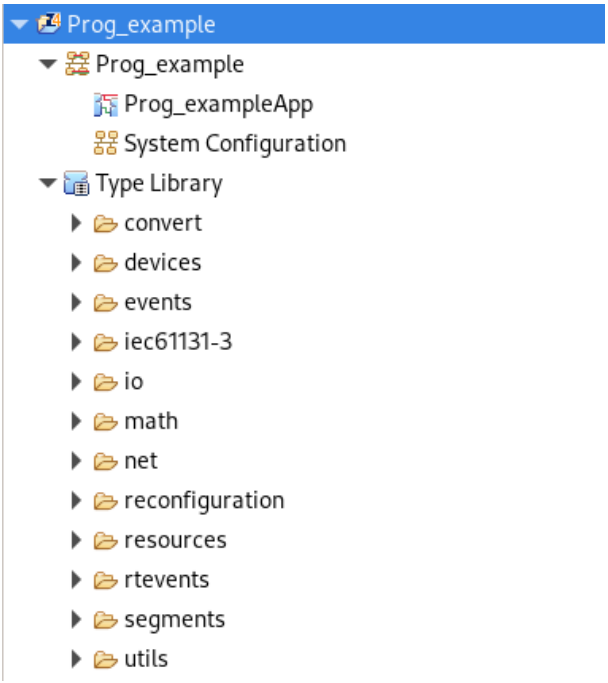

Рисунок 8.2 – Дерево проекта

Для того, чтобы загружать прошивку в ПЛК, необходимо произвести настройку параметров устройства согласно заданной конфигурации.

Для этого необходимо нажать два раза левой кнопкой мыши на *System Configuration*, после чего появится пустое поле, где с правой стороны будет меню (смотреть рисунок 8.3). В первую очередь, нам нужно выбрать устройство. Для ПЛК Ai-Talap необходимо выбрать *FORTE_PC* и перетащить его на пустое поле, как показано на рисунке 8.3. После добавления можно переименовать устройство (но это не является обязательным). В рамках данного проекта устройству присвоится имя ПЛК *plc1*. Далее, необходимо на входе *MGR_ID* указать IP-адрес ПЛК согласно конфигурационному файлу следующим образом: «*IP-addr:61499*». В рамках данного проекта запись будет выглядеть «*192.168.2.1:61499*» (смотреть рисунок 8.4). Далее нужно будет в разделе *Segments* выбрать тип сетевого канала *Ethernet* и соединить с добавленным устройством *FORTE_PC* с названием *plc1*. Также, при желании можно переименовать *EMB_RES* (программное ядро устройство, в которое в последующем будут добавляться программные блоки, само ядро необходимо для идентификации принадлежности программного кода к устройству и его компиляции для последующей загрузки). Устройство может содержать безграничное количество ядер *EMB_RES*, но в данной линейке ПЛК – пределом является 5 одновременно работающих ядер программ, а также сумма их размера не должна переполнять память *4diac Forte* в самом ПЛК.

Результат представлен на рисунке 8.4. По такому подобию можно строить распределенные системы, добавляя разное количество устройств.

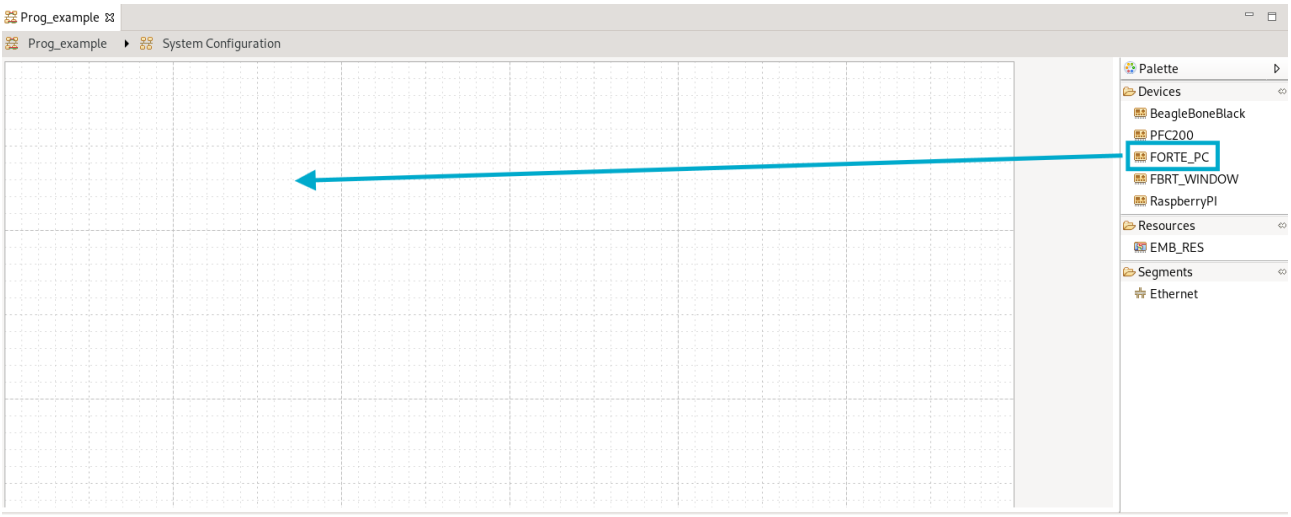

Рисунок 8.3 – Добавление устройства

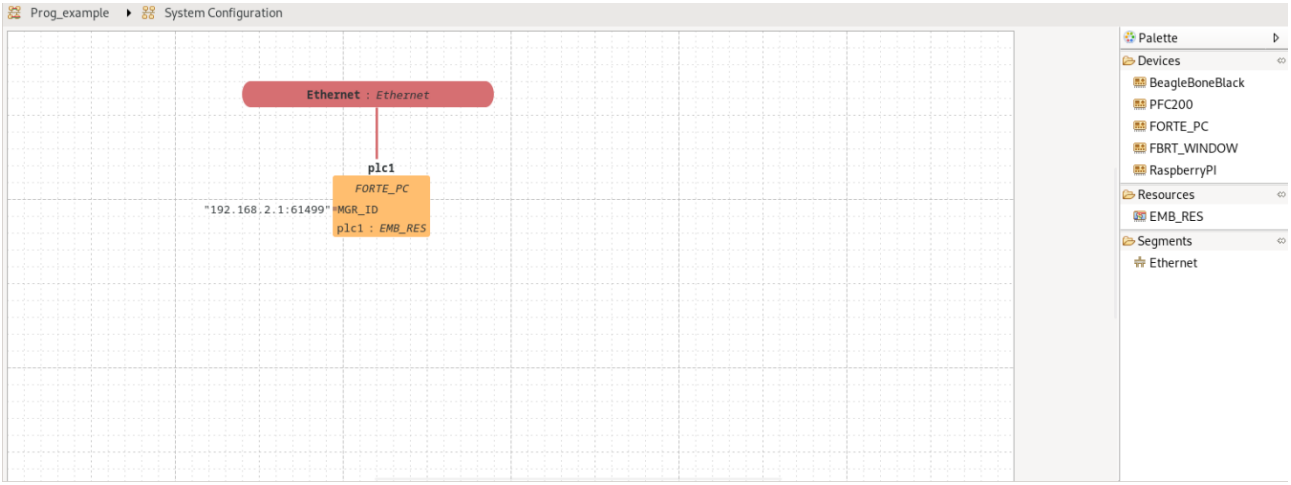

Рисунок 8.4 – Настройка устройства и добавления канала связи

8.3 Передача переменных из ПЛК в 4diac

В пункте 8.1 была рассказано про назначение *var* и *var_out* при настройке *forte*. Теперь для того, чтобы *4diac forte* их получил, необходимо использовать блок *Subscribe*. То есть, ПЛК выступает как их издатель (*var_out)*, а среда является подписчиком. Это является началом программирования. Для добавления блока необходимо два раза левой кнопкой мыши нажать на *Prog_exampleApp*, после чего появляется пустое поле. В разделе *net* с правой стороны в Type *Library* необходимо выбрать блок *SUBSCRIBE_1* (там их есть множество модификаций, но стоит помнить одно – цифра после названия говорит не о количество считывающихся переменных, а о количестве копий считанной переменной, поэтому блок *SUBSCRIBE_8* не считывает 8 переменных, а считывает одну переменную и дублирует ее на 8 выходов; на практике с одного выхода можно выводить множество линий, так что необходимость в использовании этих блоков возникает, если переменные должны иметь точную временную передачу, либо четко синхронизированы по времени, либо если нужно, чтобы код был более читабелен). Результат представлен на рисунке 8.5.

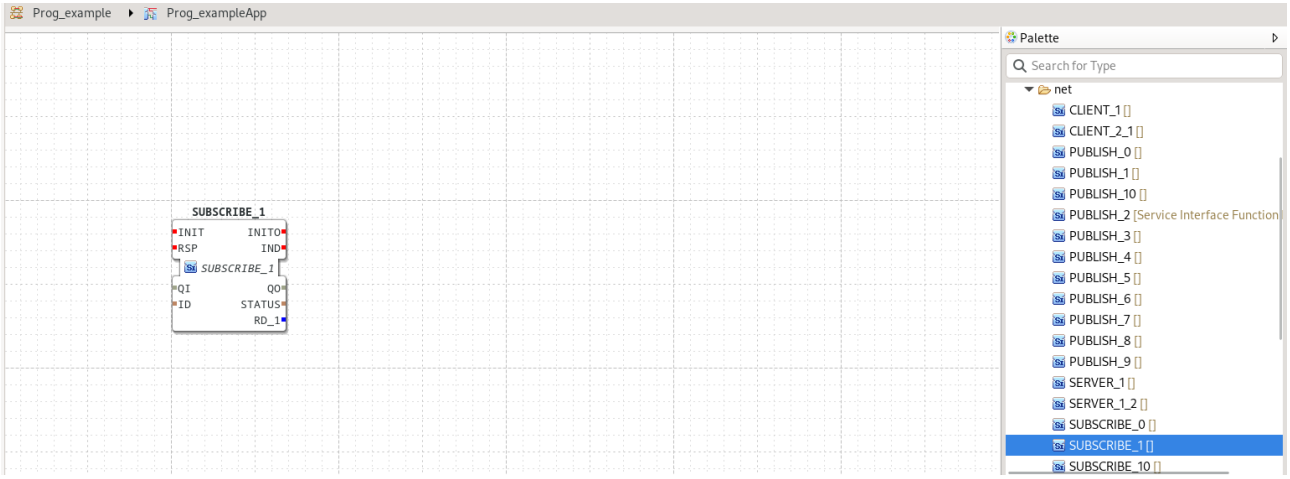

Рисунок 8.5 – Добавления блока *SUBSCRIBE_1*

Как видно из рисунка 8.5, блок имеет:

- Два входных Event-сигнала *INIT, RSP*
- Два выходных Event-сигнала *INITO, IND*
- Входной сигнал bool *QI*
- Выходной сигнал bool *QO*
- Входной сигнал WSTRING *ID*
- Выходной сигнал WSTRING *STATUS*
- Выходной сигнал ANY *RD_1*.

Входные Event сигналы необходимы для того, чтобы инициализировать работу блока (считывание данных, а также запуск работы определенных блоков) – они нужны для синхронизации выполнения программных блоков, в какой период их нужно запустить, в какие моменты отключить и т.д. В данном блоке *INIT* инициализирует работу блока, считывание переменных. *RSP* запускает в работу алгоритм блока.

Выходные Event сигналы необходимы для того, чтобы продемонстрировать, был ли отработан тот или иной блок, прошла ли инициализация и т.д. В данном блоке *INITO* показывает, был ли инициализирован блок. Идеальный случай работы блока – когда *INIT* и *INITO* меняются одновременно, либо оба одновременно находятся в состоянии изменения. *IND* показывает, отработал ли алгоритм блока.

Входной сигнал bool *QI* и Выходной сигнал bool *QO* нужны для того, чтобы организовать считывание переменных по условию. Если *QI = true*, то начинается считывания сигнала и если сигнал считан – *QO = true*.

Вход *ID* принимает наименование переменной, которую нужно принять. Для того, чтобы переменная была считана с ПЛК в этот вход необходимо задать константу в формате «*npc21[Наименование переменной]*».

Выход *STATUS* отражает текущее состояние считывания переменной.

Выход *RD_1* выдает считанное значение в формате ANY. При этом стоит обратить внимание на то, что если этот выход подать напрямую на какой-либо вход – то будет ошибка передачи данных. Перед и/или после сигнала ANY необходимо ставить конвертер в нужный формат, куда необходимо подать сигнал. Например, при считывании значения bool необходимо после ANY ставить блок *BOOL2BOOL*. Подробнее об этом будет сазано далее.

Для примера на рисунке 8.6 представлен результат настройки блока для переменной *di5*.

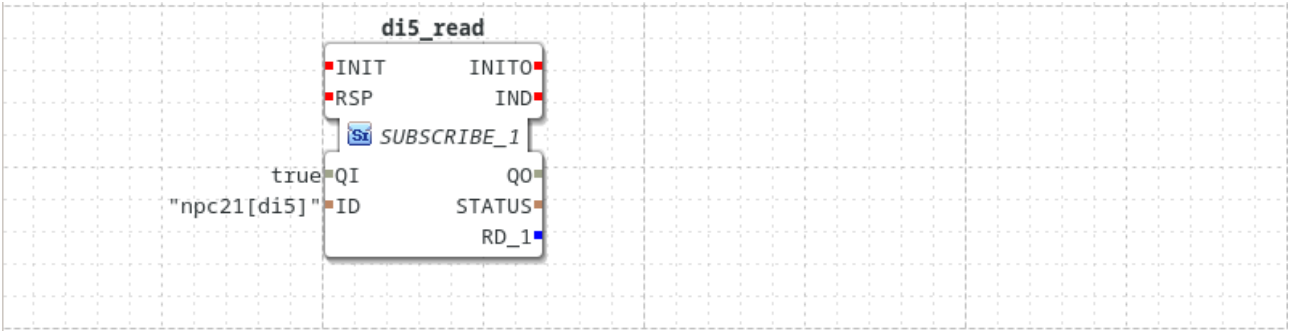

Рисунок 8.6 – Результат настройки блока для *di5*

Для остальных переменных, указанных в *var_out*, все делается аналогично. Результат представлен на рисунке 8.7

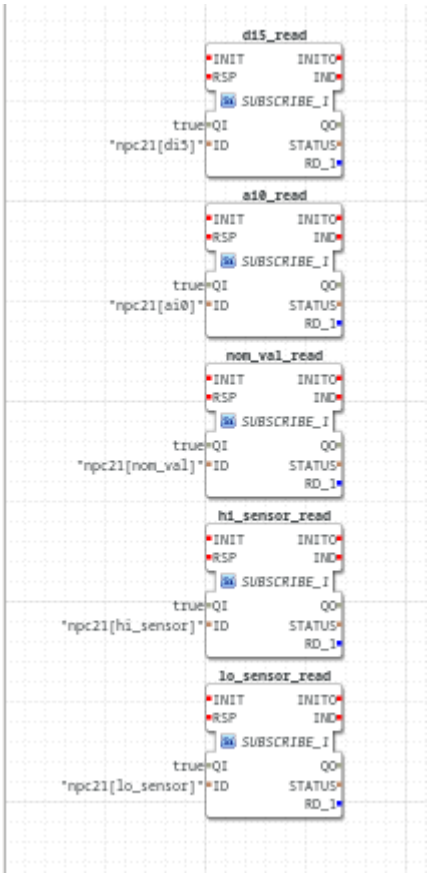

Рисунок 8.7 – Результат создание блоков *Subscribe*

8.4 Настройка цикла программы

После того, как были добавлены блоки для считывания данных, необходимо оценить, сколько может понадобиться времени, исходя из логики – они могут быть считаны одновременно, последовательно, по условию, по таймеру и т.д – это необходимо оценить, иначе можно столкнуться с проблемой, что какая-то переменная не успела считаться и ее значение на выходе будет *ND (Any)*. Также, нужно будет учесть объем программы, но на практике 100 мс хватает для большинства программ. Если же необходимо время меньше, то к этому вопросу необходимо вернуться после разработки всего ПО и его отладки. В рамках этого проекта время будет установлено 30 мс.

Для добавления цикла необходимо в правом окне в разделе *events* выбрать *E_CYCLE* (для удобства, все программы, которые связаны и зависят от Event сигналов, начинаются с *E_*). Сам блок представлен на рисунке 8.8.

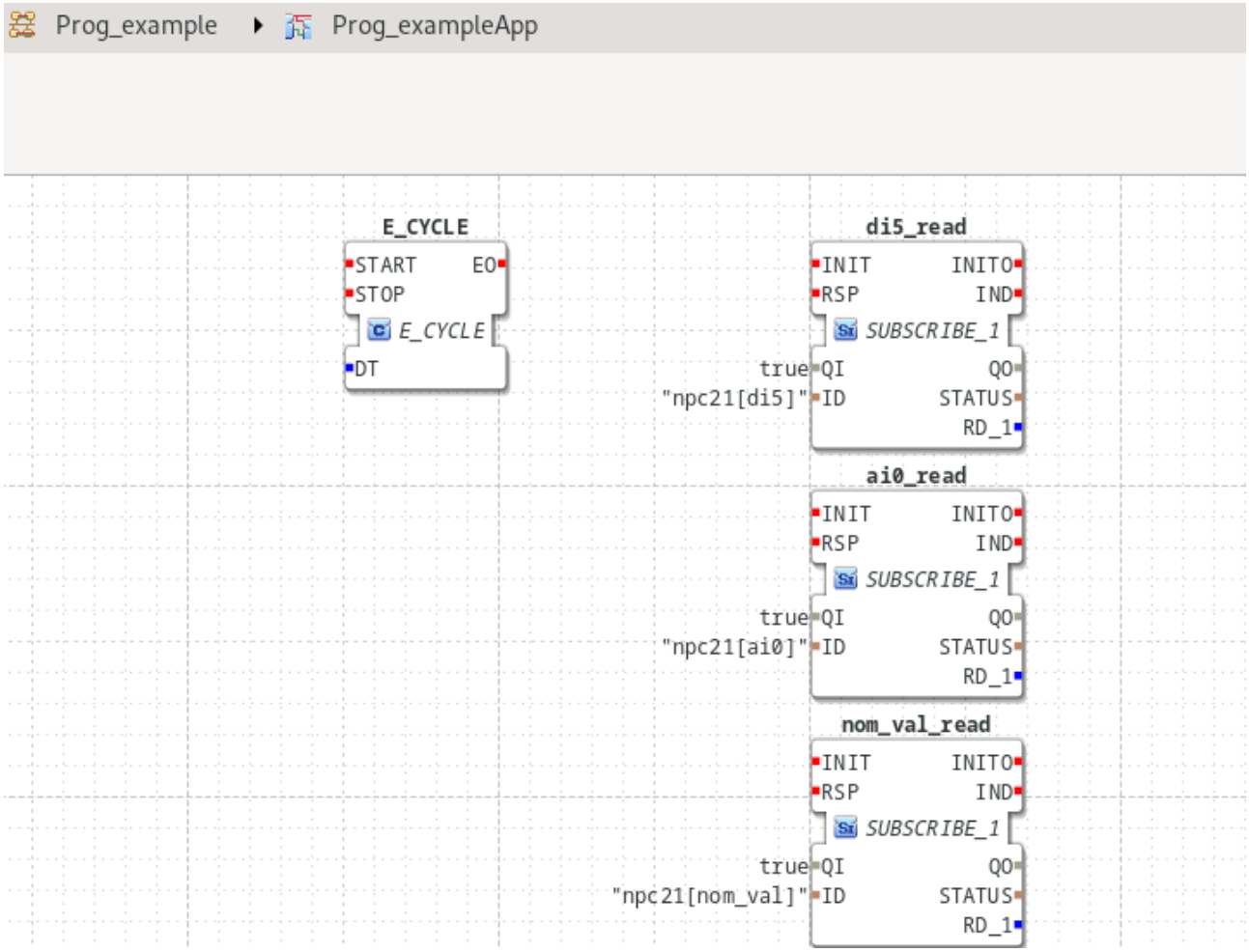

Рисунок 8.8 – Добавление блока *E_CYCLE*

Данный блок имеет два входных Event-сигнала *Start* и *Stop*. При наличии сигнала на *Start*, происходит инициализация блока и сам блок начинает генерировать в заданный интервал времени одно событие Event – в данном блоке на выходе *E0.* При поступлении сигнала на *Stop* происходит остановка генерации сигнала *E0*.

Вход *DT* имеет тип TIME и вводится в формате *T#ВремяЕдиница времени.* Для 30 мс запись будет иметь вид *T#30ms.*

Выход *E0* необходимо подключать ко входам *INIT, RSP.* Только нужно изначально знать, как они должны инициализироваться. В рамках данного проекта считается, что все переменные будут считаны одновременно, поэтому необходимо выход *E0* соединить со всеми *INIT, RSP. Start* и *Stop* будут соединены позже. Результат представлен на рисунке 8.9.

Но, иногда встречаются случаи, когда *IND* выдает сигнал события только один раз. В данном ПО данный выход является ключевым – он соединяется с другими входами событий и, если произойдет описанная ситуация, система перестанет работать. В таком случае, нужно либо ставить второй *E_CYCLE*, если критично именно считывание, который будет генерировать события только при единичном выходном событии, либо использовать выход *INITO*. В качестве примера соединения для данной программы будет использоваться выход *IND*, как пример, но на практике можно подключать и к *INITO*.

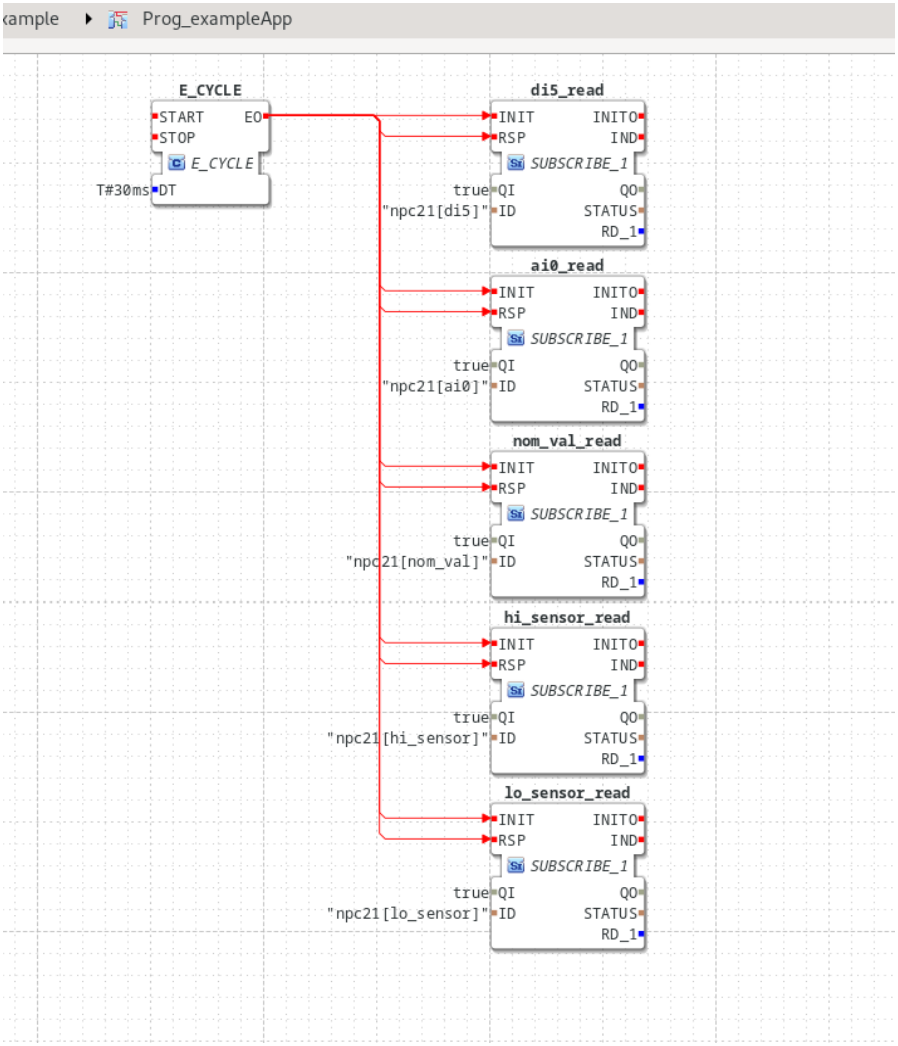

Рисунок 8.9 – Результат подключения всех блоков

8.5 Создание отдельных *Type*

Для удобства программирования функциональные блоки можно поместить в некую функцию. В среде 4diac IDE это называется типом. В зависимости от задачи существует множество типов блоков, представленных на рисунке 8.10. В данном проекте будет использоваться тип *SubApp,* который позволяет помещать функциональные блоки в отдельные функции. Для создания отдельных других типов блоков необходимо посмотреть раздел 8.12.

Стоит иметь в виду, что в данной IDE не запрещается и не ограничивается число вложенных *SubApp* в проекте, но любой добавленный блок занимает память и при попытке загрузить его в ПЛК может произойти аппаратный сброс программы и остановка ее выполнения.

Для удобства, в корневом каталоге этого проекта будет создана папка для таких блоков. Для этого необходимо нажать правой кнопкой мыши по проекту, выбрать *New->Folder* и создать папку. В этом проекте папка получила название *Custom_Sub*. Далее, для создания *SubApp* необходимо нажать правой кнопкой мыши на *Custom_Sub*, выбрать *New->Type* и в открывшемся окне выбрать *SubApp* (смотреть рисунок 8.10).

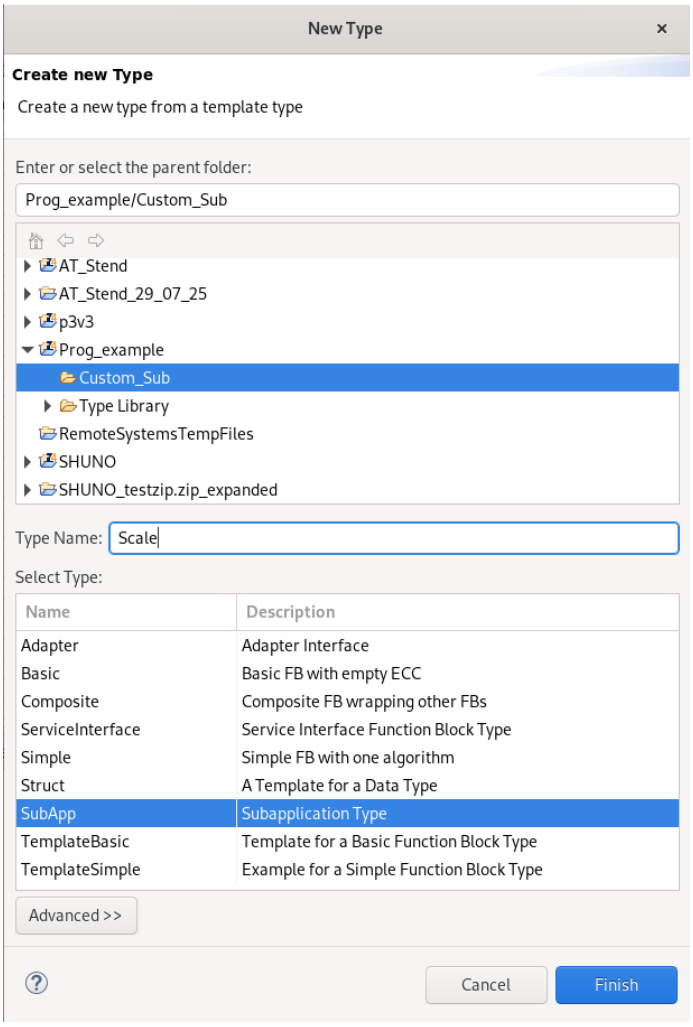

Рисунок 8.10 – Создание *SubApp*

В качестве первой функции будет создана функция масштабирования аналогового сигнала *Scale.* Результат создания и ее размещения в папке представлен на рисунке 8.11.

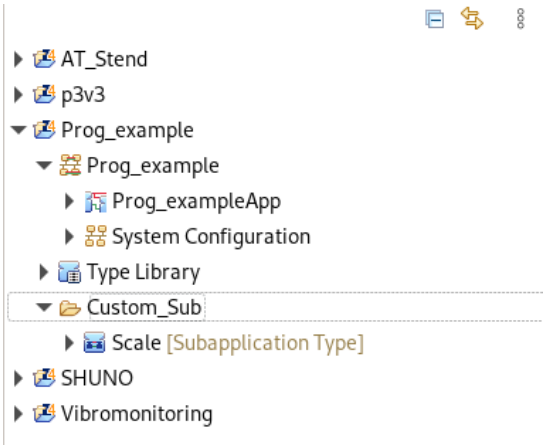

Рисунок 8.11 – Добавление блока в папку

После создания функции откроется окно редактирования и настройки блока, представленное на рисунке 8.12.

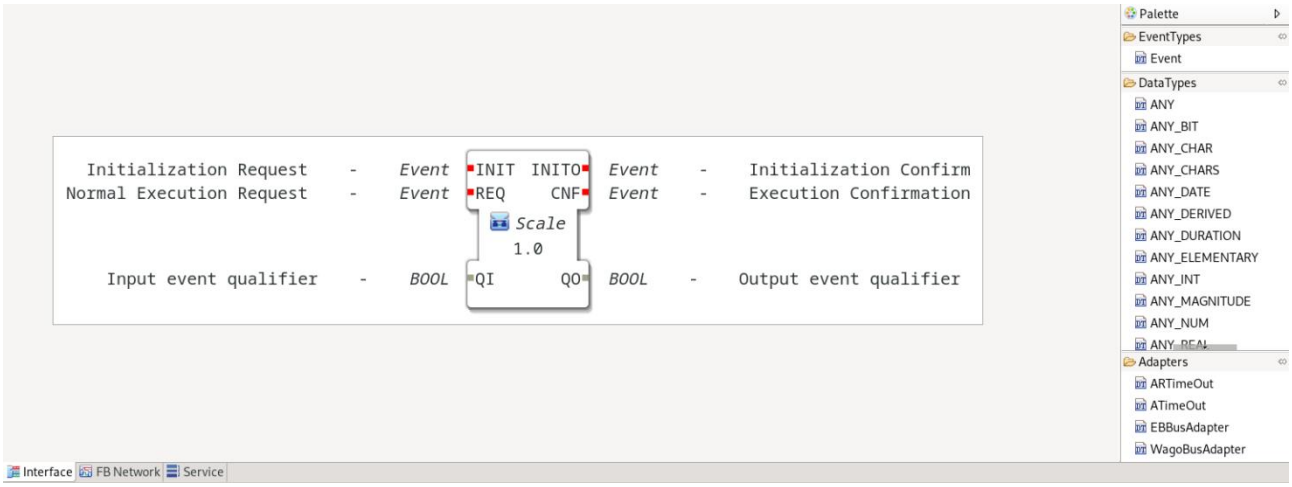

Рисунок 8.12 – Интерфейс редактирования блока

По умолчанию при создании блока создаются 3 входа *INIT, REQ, QI* и 3 выхода *INITO, CNF, QO*. При необходимости можно как добавить, так и удалить сигналы. Так как сам блок выполняет простую функцию и не планируется делать отдельно инициализацию и отдельно выполнение, то *INIT* и *INITO* можно удалить. Также *QI, QO* необходимо удалить, так как в алгоритме масштабирования они не нужны. Необходимо добавить переменные на вход: *AI* – значение аналогового сигнала, *IN_MAX* – максимальное значение входного не обработанного сигнала, в данном случае 10000, *IN_MIN* – минимальное значение входного не обработанного сигнала, в данном случае 0, *HI* – максимальное значение выходного сигнала, в данном случае 500, *LO* - минимальное значение выходного сигнала, в данном случае 0; а также переменную на выход: *ScaleSig*. Результат представлен на рисунке 8.13.

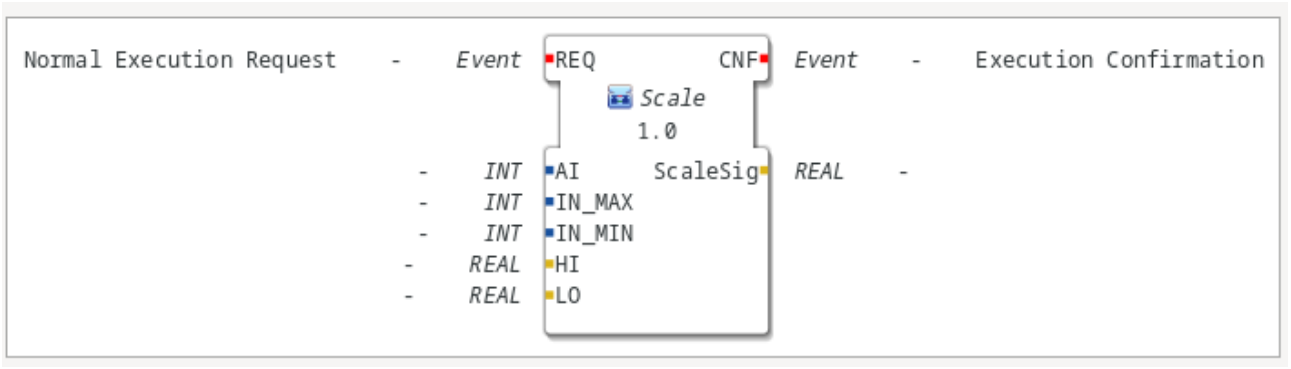

Рисунок 8.13 – Создание входов/выходов блока

Далее, для добавления блоков внутри функции необходимо перейти во вкладку *FB Network*, расположенной внизу. Окно выглядит так, как показано на рисунке 8.14.

Как видно из данного рисунка, здесь доступны как типовые блоки в папке *Type Library*, так и созданные внутри проекта, например, в папке *Custom_Sub* (за исключением самого блока, который сейчас редактируется).

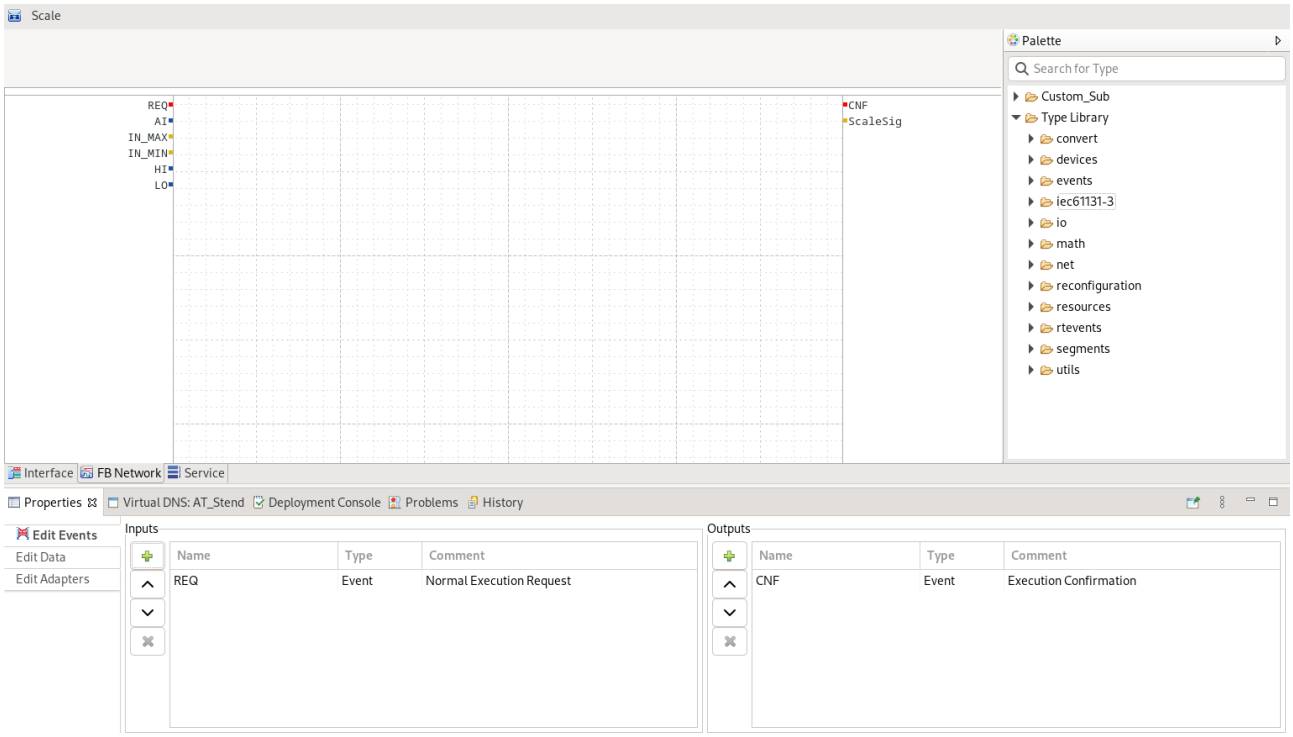

Рисунок 8.14 – Окно редактирования

Само масштабирование можно реализовать разными методами и формулами, но в рамках данного проекта будет использована стандартная формула интерполяции (8.1).

$$ScaleSig = LO + \frac{(HI - LO)}{(IN\_MAX - IN\_MIN)} \cdot (AI - IN\_MIN) \quad (8.1)$$

Где $LO, HI, IN\_MAX, IN\_MIN, AI, ScaleSig$ – ранее описанные переменные.

Для реализации этой формулы можно воспользоваться стандартными блоками из *Type Library->iec61131-3->arithmetic* *F_ADD, F_SUB, F_MUL, F_DIV* (для удобства все стандартные блоки, которые не работают с Event-сигналами, имеют префикс *F_*). Их внешний вид представлен на рисунке 8.15.

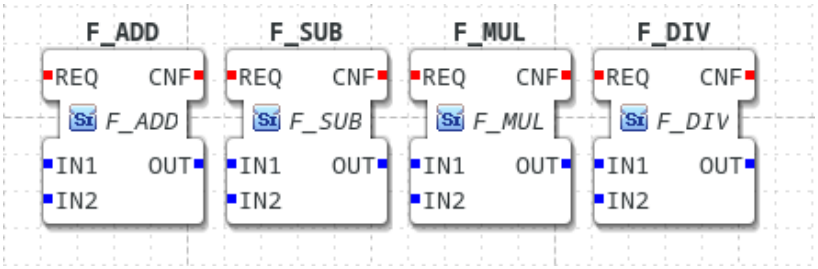

Рисунок 8.15 – Внешний вид блоков

Каждый из блоков выполняет свою математическую функцию, производя математическую операцию над входами типа *ANY IN1, IN2* и вывода результат на вход типа *ANY OUT*. События *REG, CNF* имеют тот же смысл, как и в других блоках, только здесь инициализация, чтение переменных и выполнение заданных алгоритмов производится с помощью одного Event-сигнала.

Так как тип ANY позволяет работать с любым типом переменной, кодируя ее специальном образом, то перед ними ими и после их нужно устанавливать «декодеры» сигнала в нужны формат. Они лежат в папке *Type Library->convert.* В зависимости от того, какой тип данных нужен, необходимо брать блок *ТипДанных2ТипДанных.* Например, если нужно сложить два значения типа REAL, то нужно брать блок *REAL2REAL,* если нужно использовать INT – *INT2INT,* если тип TIME – *TIME2TIME* и т.д. При этом, стоит помнить, что если на вход нужно подать константу, то на самом входе ANY пишется *ТипДанных#Число*, например, *REAL#3.5.*

В данном проекте все промежуточные операции будут иметь тип REAL.

Исходя из вышеописанного, можно заключить, что перед входами типа ANY, а также после выходов типа ANY необходимо ставить декодеры нужного формата для корректной интерпретации сигнала. Результат построения формулы (8.1) представлен на рисунке 8.16.

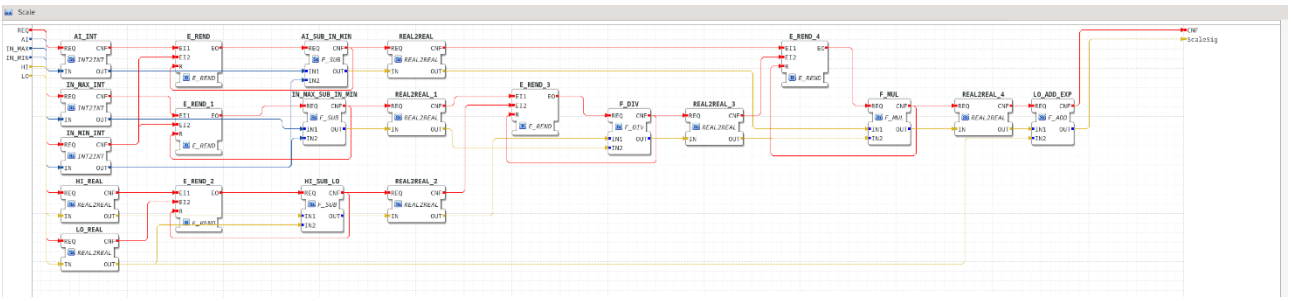

Рисунок 8.16 – Блок Scale

Как видно из данного рисунка, построение блока основывается на том же принципе, как и разработка FBD-программ. В данном *SubApp* был добавлен блок *E_REND*, из префикса которого становится понятно, что этот блок работает с Event-сигналами. Его основная задача – контроль и предотвращения перегруза ПЛК событиями. Для более детального понимания будет рассмотрен фрагмент кода, а именно *AI-IN_MIN.*

Как видно из рисунка, блок имеет один *REQ*, при этом данный блок считывает два значения – сигнал с блока *INT2INT*, который декодирует и формирует значение для переменной *AI (блок AI_INT)* и *IN_MIN (*блок *IN_MIN_INT)*, каждая из которых имеет свой выход *CNF*. Среда 4diac forte не запрещает устанавливать на один вход два сигнала Event и в простых случаях это считается более правильным способом для оптимизации расчетов, но сама по себе структура такого программирования имеет ряд проблем и ограничений.

Первое, как было сказано, Event-сигналы имеют одно назначение – управлять поведением блоков и ходом выполнения ПО. Если на один вход ставить два сигнал Event, то блок запустится при сигнале одного из двух сигналов блоков. Если важно использовать именно выходные значения каждого блока, то такой подход приведет к потере точности. Эта ситуация может выглядеть следующим образом.

Допустим, блока *AI_INT* и *IN_MIN_INT* в первом цикле прошли инициализацию одновременно и получены значения 5000 и 0 соответственно. Выход с блока *AI_SUB_IN_MIN* будет равен 5 000. А во втором цикле произошла задержка и блок *AI_INT* не успел инициализироваться, и его значение, равное 6 000, не вышло на выход, и по правила «событийного» программирования на выходе останется прежнее значение 5000. Тогда, блок *AI_SUB_IN_MIN* отработает, выдаст сигнал *CNF*, что означает его успешное выполнение, а результат разности вместо 6 000 покажет 5000.

Как видно, для этой программы это не имеет смысла, ведь даже сам блок не отработает и значение так и будет 5 000. **НО!!!** Опасность заключается не только в неточности результата, а в том, что программой будет считаться успешность выполнения кода и *CNF* блока запустит следующий блок в работу. А за этим могут идти ошибки синхронизации, ошибки поведения

программы и запуск тех фрагментов программы, что не должны работать. Как уже говорилось ранее, для простых программ это не имеет значения, основной риск для распределенных систем, где важно точность и полная синхронизация.

Вторая проблема, это увеличение разности между выходным сигналом Event при запуске исполнения программы (в данном случае, выход с *E_CYCLE*) и выходом Event в самом последнем блоке прикладной программы, которая все завершает (для любого проекта, по большей части, это будет блок *Publish*, который будет передавать значения в ПЛК). Сама IDE не имеет никаких ограничений, а вот исполнительная среда forte внутри ПЛК имеет ограничения. При разработке учитывался максимально возможный «завал» событий для повышенной устойчивости перегруза ПЛК, но в рамках тестирования выявлено, что при разнице в 101 000 событий ПЛК останавливает выполнение программы. **НО!!!** Опасность все так же по большей части заключается в рассогласованности работы распределенной системы и их ошибках в синхронизации.

Третья проблема больше носит описательный характер. При таком подходе FBD-код плохо читается и не всегда сразу получается найти источник событий и код становится «грязным» для восприятия.

В самой среде для избежания этой проблемы есть стандартные блоки *E_REND* и *E_MERGE.* Их вид представлен на рисунке 8.17.

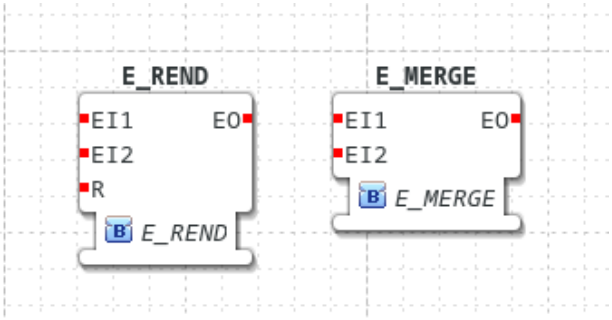

Рисунок 8.17 – Блоки *E_REND* и *E_MERGE.*

Для необходимы для синхронизации и фильтрации событий и регулировки количества поступаемых событий. Простыми словами, они являются аналогами функций *И* и *ИЛИ.*

В зависимости от задачи выбирается логика управления событиями. Если инициализация конечного блока должна происходить по одному из блоку, чьи сигналы поступают на него, тогда выбирается *E_MERGE.* В своем исполнении он прост. Если поступают события хотя бы одного блока (*EI1* или *EI2*), то на выходе *EO* появляется выход события.

Если нужно, чтобы инициализация проходила только, если два блока выдали событие, то необходим использовать *E_REND*. Он работает следующим образом. Если есть события с двух блоков одновременно (*EI1* и *EI2*), то на выходе *EO* появляется выход события. При этом, чтобы сравнение было возобновлено (иначе выход застынет в одном положении), необходимо установить триггер сброса *R*. В данном проекте триггером выступает успешная отработка блока (выход *CNF* с блока, на чей *REQ* поступил *EO*), который был запущен через *EO* на подобие обратной связи.

Исходя из задачи (вычисления значений), оба значения должны быть синхронны, значит необходимо использовать блок *E_REND*. Принцип его использования представлен на рисунке 8.16.

Другие блоки создаются аналогично. Результат создания блока для контроля температуры *Check_temp* представлен на рисунках 8.18 и 8.19.

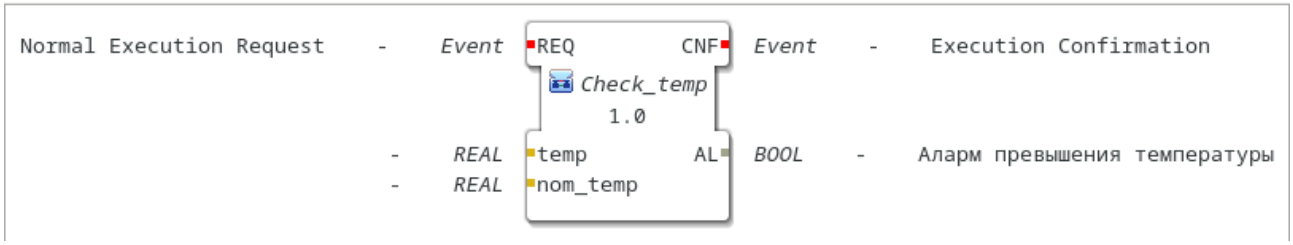

Рисунок 8.18 – Создание интерфейса входов/выходов блока

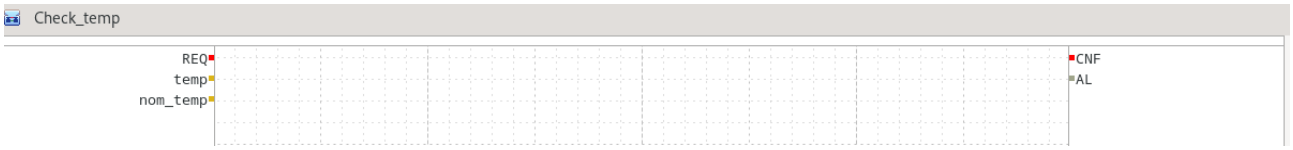

Рисунок 8.19 – Пустая форма редактирования во вкладке *FB Network*

Для сравнения можно использовать специальные встроенные библиотечные блоки, которые лежат в *Type Library->iec61131-3->comparison*. Для данного проекта будет использоваться блок «больше чем» *F_GT*. Его вид представлен на рисунке 8.20.

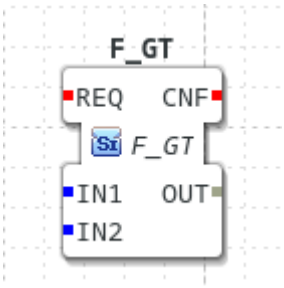

Рисунок 8.20 – Блок *F_GT*

Он имеет по одному сигналу Event для входа и выхода, два входных сигнала типа ANY *IN1, IN2* и один выходной сигнал типа BOOL *OUT*. Алгоритм его работы прост. При инициализации блока (поступления сигнала на *REQ*) происходит сравнение *IN1>IN2*. Результат сравнения выводится в переменную *OUT*.

В рамках данной задачи использование блока было не обязательным, но при программировании реальных процессов стоит иметь в виду, что лучше всего использовать отдельные блоки по функциональному признаку, даже если их всего два-три внутри. Помимо читабельности, универсальности и возможности изменения алгоритмов блоков, это позволяет избежать путаницы в управлении и добавлении лишних *E_MERGE* и *E_REND*.

Также, рекомендуется помещать блоки считывания данных в один блок и комбинировать их вместе с *E_MERGE* и *E_REND*, чтобы в главном окне избежать этих блоков.

Результат разработки блока *Check_temp* представлен на рисунке 8.21.

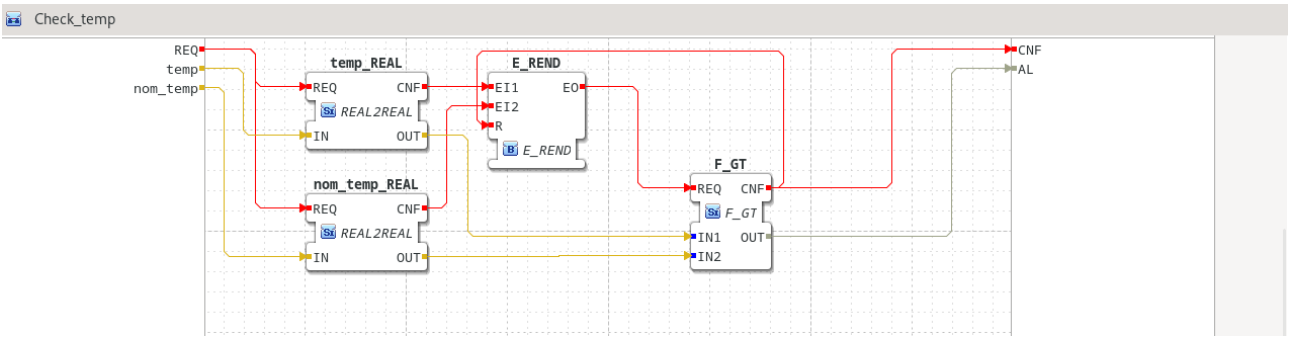

Рисунок 8.21 – Разработка блока *Check_temp*

Далее, следует сразу разобрать блок *UnScale*. Исходя из простых математических законов, для масштабирования в обратную сторону можно использовать этот же блок, задавая разные значения, но, во-первых, здесь будут разные типы данных, во-вторых, для лучшей читабельности лучше разделять функции процессов. Для блока *UnScale* формула (8.1) примет вид (8.2).

$$UnScaleSig = OUT\_MIN + \frac{(OUT\_MAX - OUT\_MIN)}{(HI - LO)} \cdot (Num - LO) \qquad (8.2)$$

Где Num – число, которое нужно преобразовать;
OUT_MAX, OUT_MIN – шкала выходного сигнала, в данном случае от 100 до 0;
HI, LO – диапазоны датчика.

Для того, чтобы заново не создавать данный тип блока, можно скопировать его и изменить. Для этого необходимо в левом окне (дерево проекта), выбрать нужный блок, который нужно скопировать, нажать правую кнопку мыши и нажать на *Copy*, либо просто выбрать блок и нажать комбинацию *CTRL* и *C*. Затем нажать на папку, где мы хоти сохранить блок, затем правая кнопка мыши, после *Paste*, либо просто *CTRL* и *V*. После этого вылезет окно, показанное на рисунке 8.22, где можно ввести новое имя блока.

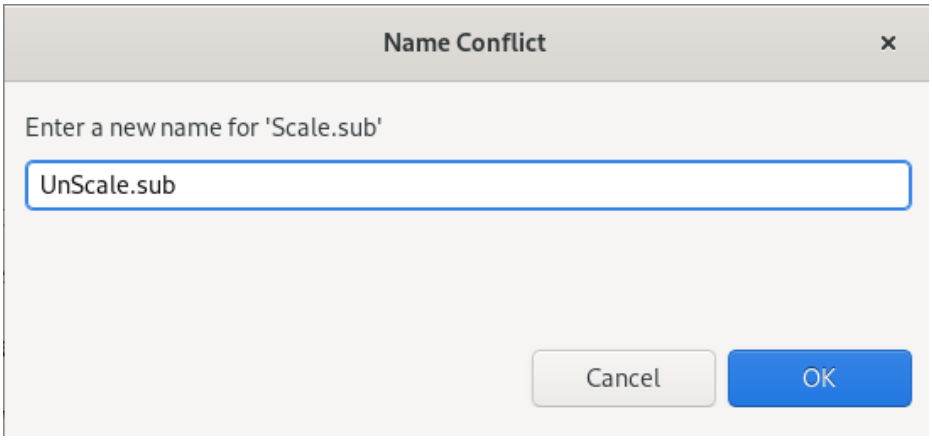

Рисунок 8.22 – Окно изменения названия

Результат в дереве проекта представлен на рисунке 8.23.

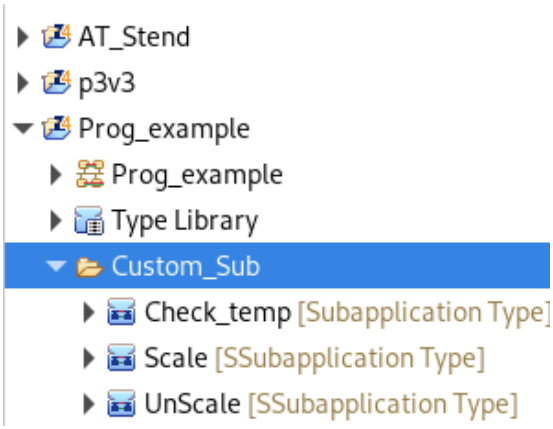

Рисунок 8.23 – Дерево проекта

Теперь достаточно изменить названия и тип входов, далее перепроверить декодеры для типов данных.

Результат изменения интерфейса входов (вкладка *Interface)* представлен на рисунке 8.24.

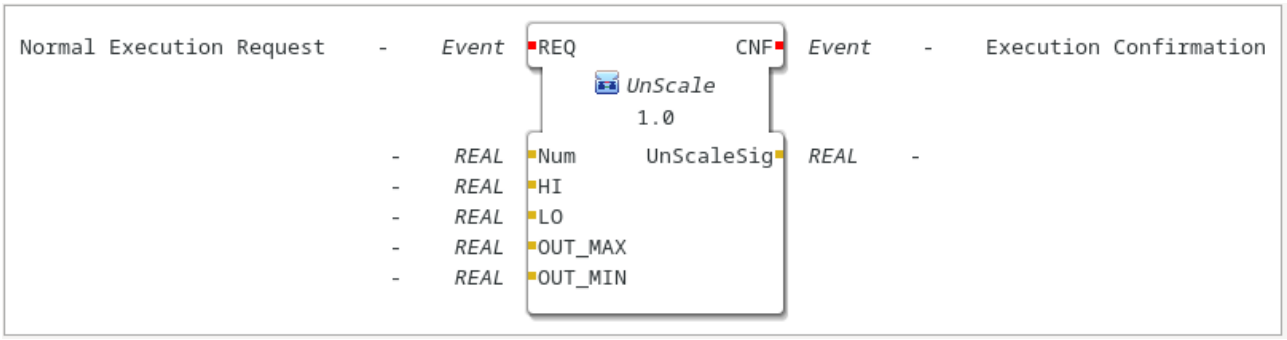

Рисунок 8.24 – Интерфейс блока

Для того, чтобы изменить блок, можно либо удалить его и добавить новый, либо воспользоваться встроенной функцией замены блока. Для этого выбрать блок, нажать правую кнопку мыши и нажать *Change Type* (смотреть рисунок 8.25). После этого появится окно, где можно либо выбрать, либо ввести в поиск и выбрать нужный тип блока. В данном случае нужно выбрать *REAL2REAL* (смотреть рисунок 8.26). **Важно!!!** Перед тем, как выполнять такие функции, да и любые, которые связаны с изменением блока, необходимо все сохранять, потому что бывают моменты, когда среда 4diac зависает. Иногда, такую проблему можно решить на Linux (тестировалось в среде Debian) путем нажатия кнопки «Перезапуск», потом, когда вылезет окно «*Выполнить перезагрузку*», нажать отмена, чтобы в самом деле не перезагрузить систему, далее на предложение сохранить проект перед выходом нужно нажать *Cancel*.

Результат изменений типов декодеров (и их названия для улучшения читабельности, что можно избегать для ускорения разработки, так как менять названия экземпляров не обязательно, да и не нужно), представлен на рисунке 8.27.

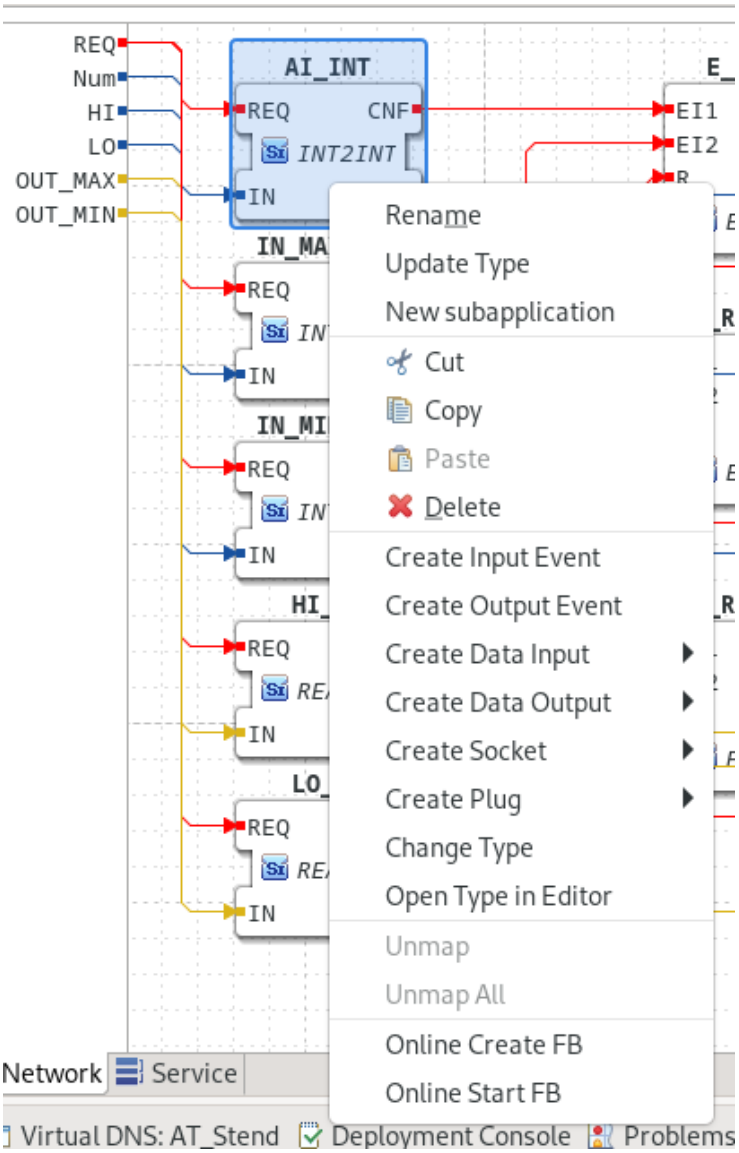

Рисунок 8.25 – Выбор функции замены блока

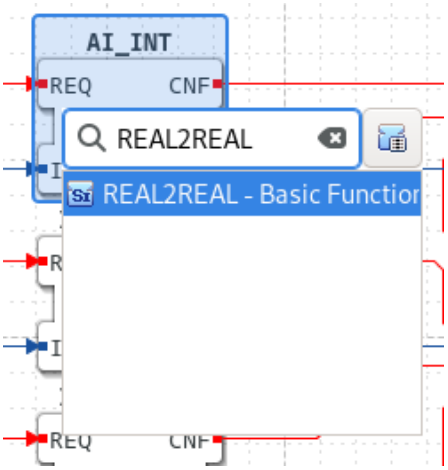

Рисунок 8.26 – Выбор нужной функции

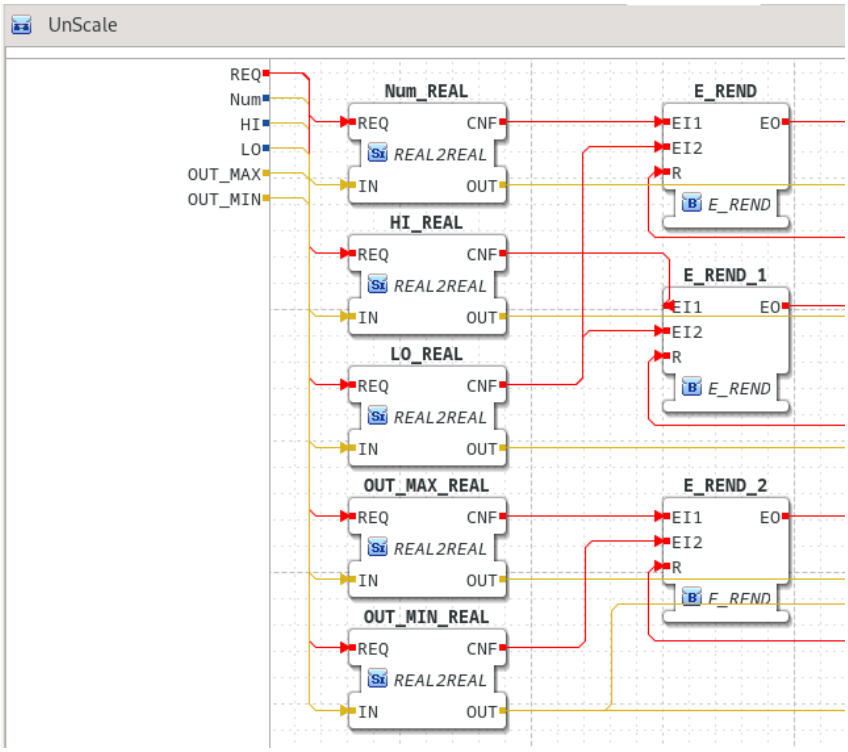

Рисунок 8.27 – Изменение блоков

Как видно из рисунка, были изменены только блоки преобразования (конвертирования). Также, стоит обратить внимание на то, что первые три сигнала (начиная с *Num*) остались синими цветом. Если они соединились с блоком *REAL2REAL* и это не вызвало ошибки, значит, это системный глюк отображения, который встречается в 100% случаях при изменении типа данных. После закрытия блока и его повторного открытия ошибка исчезает (смотреть рисунок 8.28).

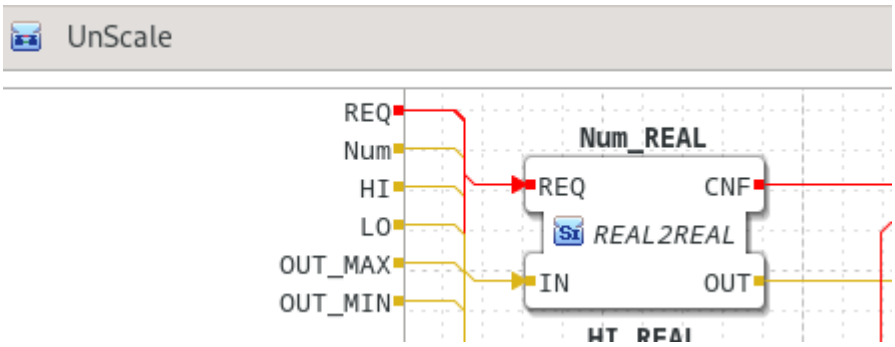

Рисунок 8.28 – Правильное отображение данных

После создания всех блоков их необходимо собрать в единый проект.

8.6 Добавление блоков в главной программе

Добавление блоков в головном программном модуле *Prog_exampleApp* выполняется по тем же принципам, как и добавление блоков в разделе 8.5. Результат добавления блоков, выполняющих функцию согласно заданному алгоритму, представлена на рисунке 8.29.

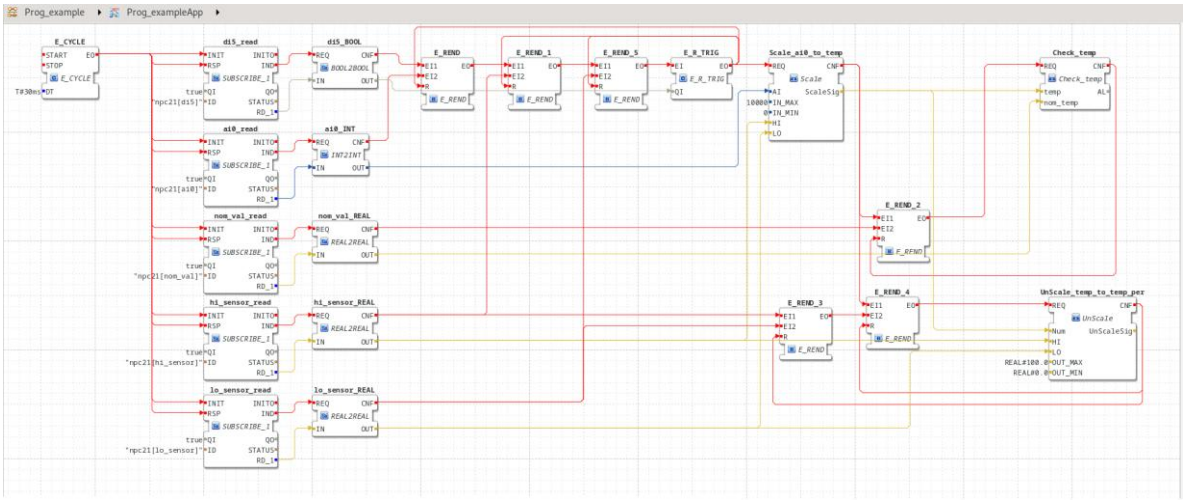

Рисунок 8.29 – Добавление блоков в *Prog_exampleApp*

На данном рисунке можно обнаружить блок *E_R_TRIG*. Он позволяет запускать события *EO* при наличии сигнала TRUE на входе *QI*. Только при использовании важно помнить – он работает по возрастающему фронту сигнала. Даже если TRUE на входе *QI* постоянно, то выход *EO* не будет изменяться до тех пор, пока вход *QI* не станет FALSE, затем снова TRUE. Подобный блок, только для падающего фронта имеет название *E_F_TRIG*. Оба этих блока лежат в *Type Library->events*. Их вид представлен на рисунке 8.30. Исходя из вышеописанного, стоит ожидать, что работа всех блоков, инициализация которых зависит от триггеров событий, переходит из режима реального времени в режим ожидания события *QI*.

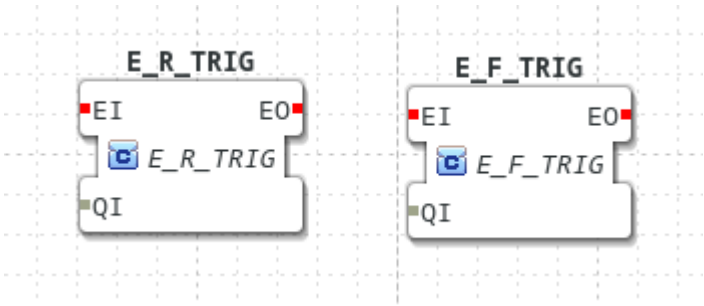

Рисунок 8.30 – Блоки *E_R_TRIG* и *E_F_TRIG*

8.7 Передача переменных в ПЛК из 4diac

В дополнение к разделу 8.3, при передаче данных в ПЛК из 4diac структура взаимодействия остается такой же, поэтому для их отправки необходимо использовать блок *Publish*. К нему применяются все те же правила наименования и правила выбора, что для *Subscribe*. Находится он в том же месте: *Type Library->net.* В рамках данной задачи будет выбран блок *PUBLISH_1*. Его вид представлен на рисунке 8.31. Он имеет все те же переменные и их описание, ту же функциональную роль, только в обратном порядке. Использование переменных остается таким же, как и в блоке *SUBSCRIBE_1.* Единственное отличие – сигнал *SD_1* в этом блоке является входом, а не выходом. На него подключают сигнал, который необходимо передать и запишется в ту переменную, которая указан в *ID* в формате «*npc21[Наименование переменной]*».

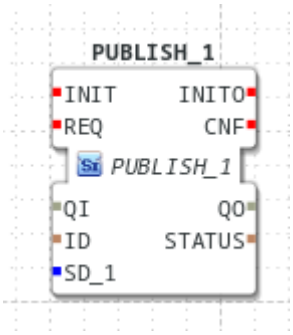

Рисунок 8.31 – Блок *PUBLISH_1*

Для всех переменных, указанных в разделе *var* в конфигурации, необходимо добавить блок *PUBLISH_1* и соединить с нужными сигналами. Результат представлен на рисунке 8.32.

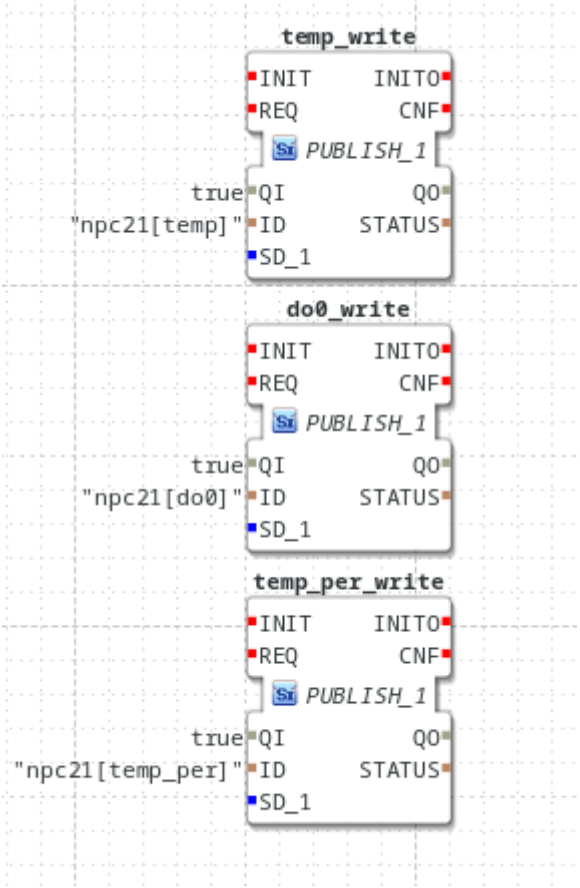

Рисунок 8.32 – Создание блоков *PUBLISH_1*

Как видно из данного рисунка, блок *PUBLISH_1* практически идентичен блоку *SUBSCRIBE_1*, за исключением некоторых особенностей, описанных раньше. По поводу подключения сигнала типа Event стоит отметить следующее. В данном случае нам нужна не синхронная отправка значений в ПЛК, а по мере их готовности. В этом случае, для инициализации достаточно их связать с тем блоком, который является конечным для формирования выходного значения.

На рисунке 8.33 представлена полная программа со всеми подключениями.

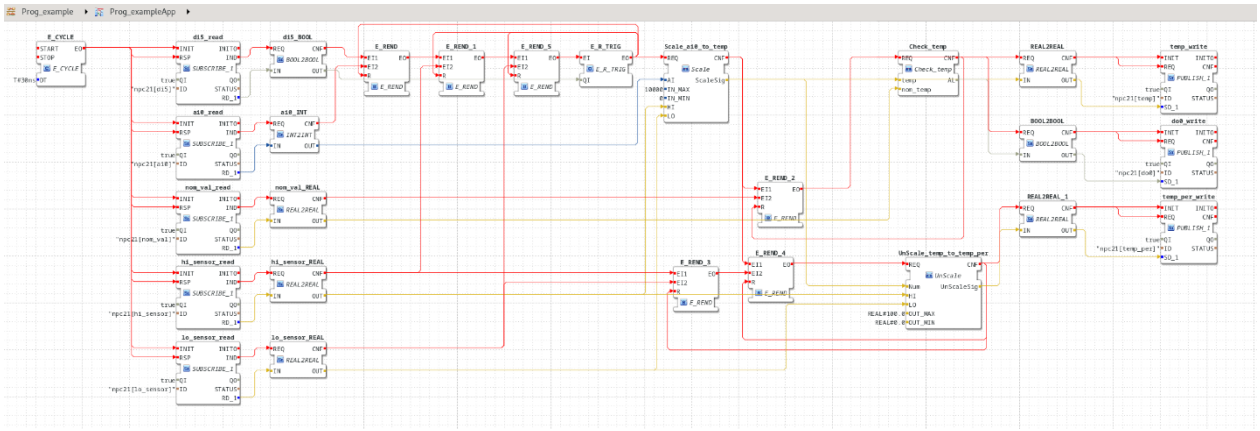

Рисунок 8.33 – Полная схема программы ПЛК

8.8 Пересоздание экземпляра блока

Зачастую при проведении комплексного тестирования выявляется ряд замечаний, которые требуют исправления в созданном блоке. После внесения изменений следует обновить экземпляр блока, в противном случае останется та же исполняемая программа. Также стоит иметь в виду, что при изменении блока меняется весь данный тип, а не конкретно этот блок в программе. Это стоит учитывать при проектировании. Если же необходимо изменить только один конкретный блок, в таком случае необходимо произвести отвязку от родительского блока. Для подробной информации смотреть раздел 8.9.

Для рассмотрения примера изменения блока будет взять блок *Check_temp*. Предположим, что появилась возможность производить подсчет количества случаев превышения температуры. В таком случае необходимо добавить счетчик.

В первую очередь необходимо добавить выходной сигнал *Count* такими же способами, как это было описано в разделе в 8.5. Для этого можно либо зайти в редактирование блока через дерево проекта, либо выбрать блок на схеме, нажать правую кнопку мыши и выбрать *Open Type Editor*. Результат представлен на рисунке 8.34.

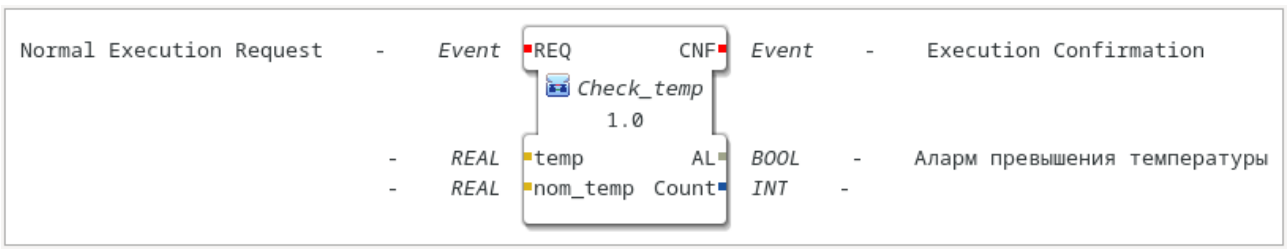

Рисунок 8.34 – Добавление входного сигнала

Далее нужно добавить счетчик. Для этого в стандартной библиотеке в *Type Library->iec61131-3->counters* есть счетчики разных видов. Самые основные *FB_CTD* (уменьшение счета) и *FB_CTU* (увеличение счета). Для этой задачи подойдет блок *FB_CTU*. **НО!!!** Стоит иметь в виду, что данный блок отличается от стандартных блоков Siemens, Schneider и других – он будет вести подсчет ровно до тех пор, пока на его входе есть сигнал, о чем будет сказано далее.

Блоки *FB_CTD* и *FB_CTU* имеют вид, представленный на рисунке 8.35. Они оба имеют одинаковые (по функциям) входы/выходы: *REQ, CNF* – сигналы Event для инициализации, *CD (CU)* – для обозначения триггера уменьшения (увеличения) счета, *LD (R)* – для загрузки (сброса) значения до *PV*, *PV* – стартовое значение, Q – выход (при наличии счета – true), *CV* – результат подсчета. Также, необходимо напомнить, что счет *PV* в сторону увеличения или

уменьшения идет при наличии сигнала от *CD (CU)*. Отсюда следует, что для корректной работы счетчиков нужно брат блоки для обнаружения фронта сигнала. Они лежат в *Type Library->iec61131-3->edgeDetection* и называются *FB_F_TRIG* и *FB_R_TRIG*. Первый определяет «падающий фронт сигнала» (когда происходит изменение с 1 на 0), а второй – «растущий» (когда происходит изменение с 0 на 1). Их вид представлен на рисунке 8.36. Они имеют одинаковые входы/выходы: *REQ, CNF* – сигналы Event для инициализации, *CLK* – сигнал, чей фронт-определяют, *Q* – результат определения фронта.

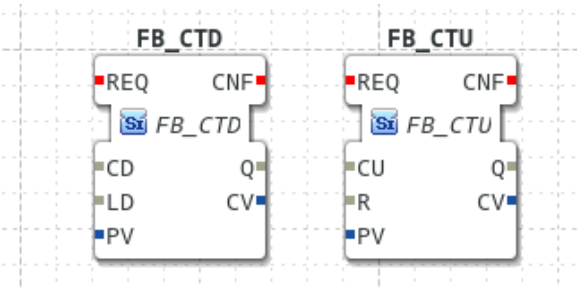

Рисунок 8.35 – Внешний вид блоков *FB_CTD* и *FB_CTU*

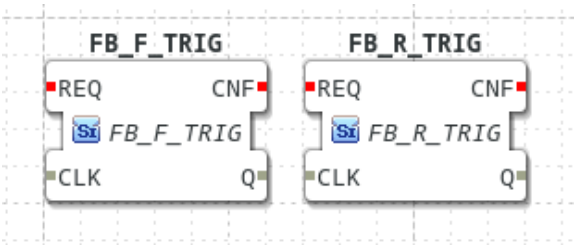

Рисунок 8.36 – Внешний вид блоков *FB_F_TRIG* и *FB_R_TRIG*

В данной случае лучше воспользоваться именно «растущим» фронтом, так как нужно именно определение аварии (когда переменная авария переходит из 0 в 1). Результат изменения блока представлен на рисунке 8.37.

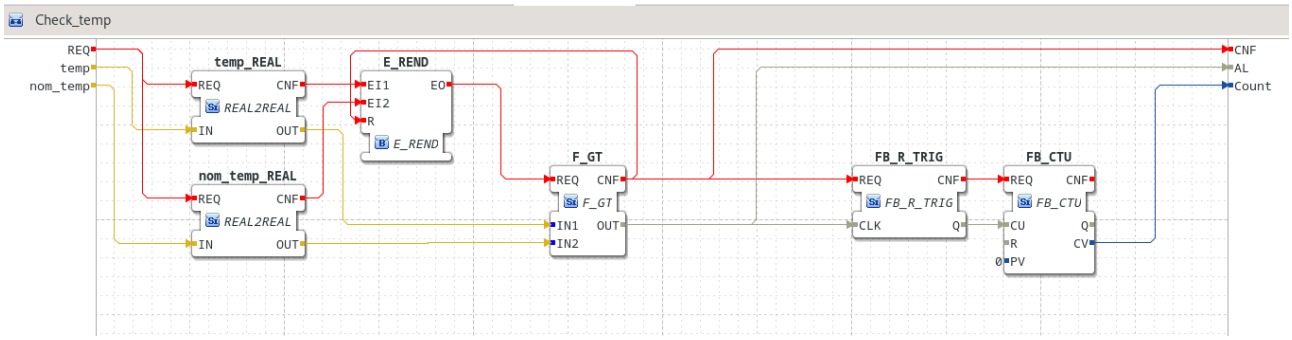

Рисунок 8.37 – Изменение программы

В качестве примечания хочется отметить, что в данном случае условия сбрасывания счетчика не предусматривались, а также вся логика была построена на том, что сигнал аварии является ключевым в выполнении, а значит работа всего блока должна зависеть именно от этого сигнала, поэтому выход *CNF* разрабатываемой функции соединен с выходом *CNF* блока *F_GT*, а не блоком *FB_CTU* (но стоит помнить, что подобные решения могут приводить к рассинхронизации и если количество блоков после *F_GT* будет большим, то это может привести к внутреннему зависанию блока). Поэтому, стоит помнить о том, что при событийном

программировании у конечного программиста больше возможностей управлять программой, при этом важно учитывать любое событие и его следствие его выполнения для всей программы.

После необходимо сохранить блок и перейти в основную программу (для напоминания, она находится *Prog_example-> Prog_example-> Prog_exampleApp*. Далее, выбрать блок *Check_temp*, нажать правую кнопку мыши и выбрать *Update Type*. Результат представлен на рисунках 8.38 и 8.39.

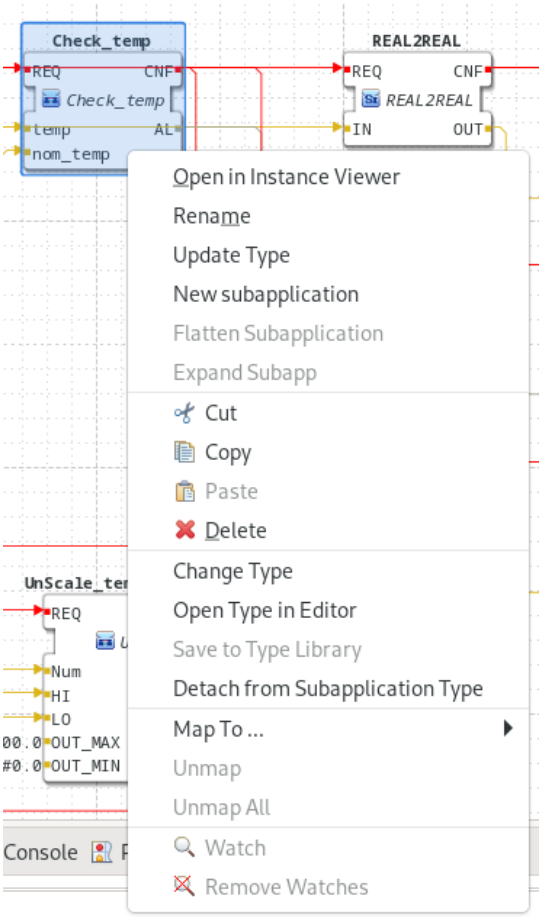

Рисунок 8.38 – Выбор обновления типа

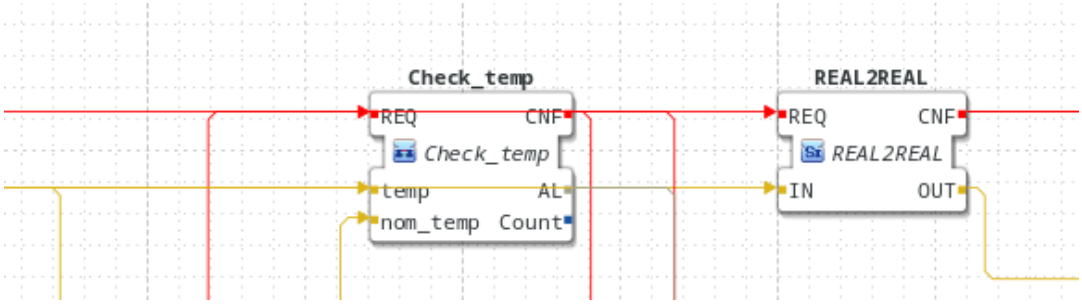

Рисунок 8.39 – Результат обновления блока

Также, если необходимо все вернуть – алгоритм тот же: изменение программы, сохранение, обновление типа.

8.9 Изменение только одного экземпляра блока

Бывают ситуации, когда необходимо поменять определенный тип блока, при этом конкретно в этой программе и в определенном месте. Только стоит помнить – если изменения требуют добавления или удаления входов/выходов, то такой метод не сработает – так как один экземпляр открепляется от родителя и его созданный образ позволяет только менять внутреннюю часть кода, но не интерфейсы. Также стоит помнить, что это работает только для блоков типа *Subapplication Type*.

В качестве примера будет рассмотрен тот же блок *Check_temp*, без примера изменений самого кода, только алгоритм отвязки.

Нужно выделить блок, нажать правую кнопку мыши (смотреть рисунок 8.40) и выбрать *Detach from Subapplication Type*.

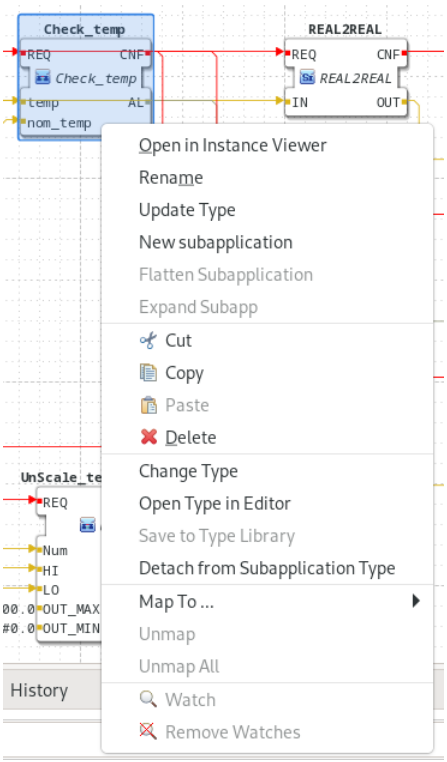

Рисунок 8.40 – Отделение от родительского блока

После отделения блок будет выглядеть так, как показано на рисунке 8.41

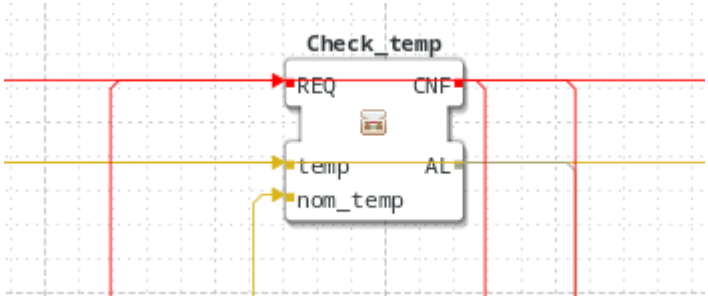

Рисунок 8.41 – Результат отделения от родительского блока

Далее, можно нажать правой кнопкой мыши, выбрать *Open* и менять код внутри (смотреть рисунок 8.42).

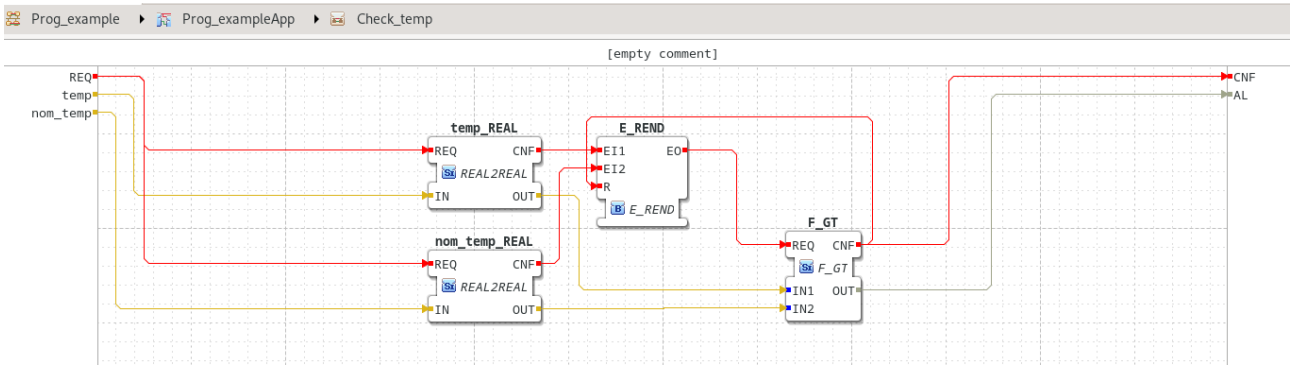

Рисунок 8.42 – Редактирование блока

8.10 Форматирование программы и ее загрузка в режиме онлайн

После того, как программа разработана, необходимо присвоить каждый блок определенному устройству и его ядру. О наличие созданных устройств и его ядер можно узнать из дерева проекта *Prog_example-> Prog_example->System Configuration.* Также, если раскрыть устройства, то можно увидеть их ядра. Пример представлен на рисунке 8.43.

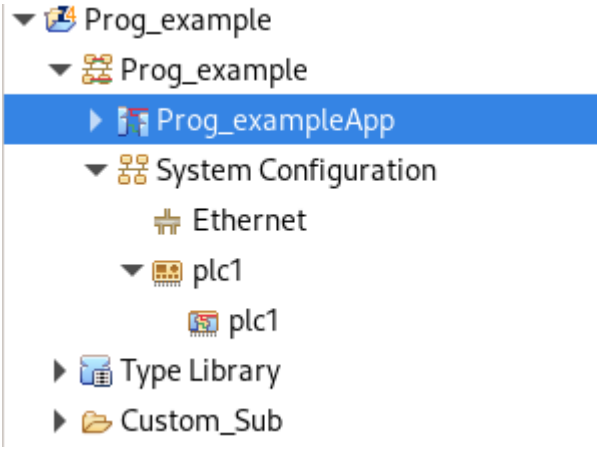

Рисунок 8.43 – Наличие устройств

Это крайне важный этап построения программного обеспечения, так как в зависимости от архитектуры управления можно один *App* (в данном проекте *Prog_exampleApp*) разбить на несколько устройств, а можно на каждое устройство создать свой *App* (для этого нужно выбрать *Prog_example,* правая кнопка мыши, затем *New->Application).*

В рамках данного проекта считается, что один *App* предназначен для одного ядра одного устройства.

Для того, чтобы добавить весь код в ядро, необходимо его выделить (*Ctrl+A)*, нажать правую кнопку мыши, затем *Map To…->plc1->plc1*. Результат представлен на рисунке 8.44 и 8.45.

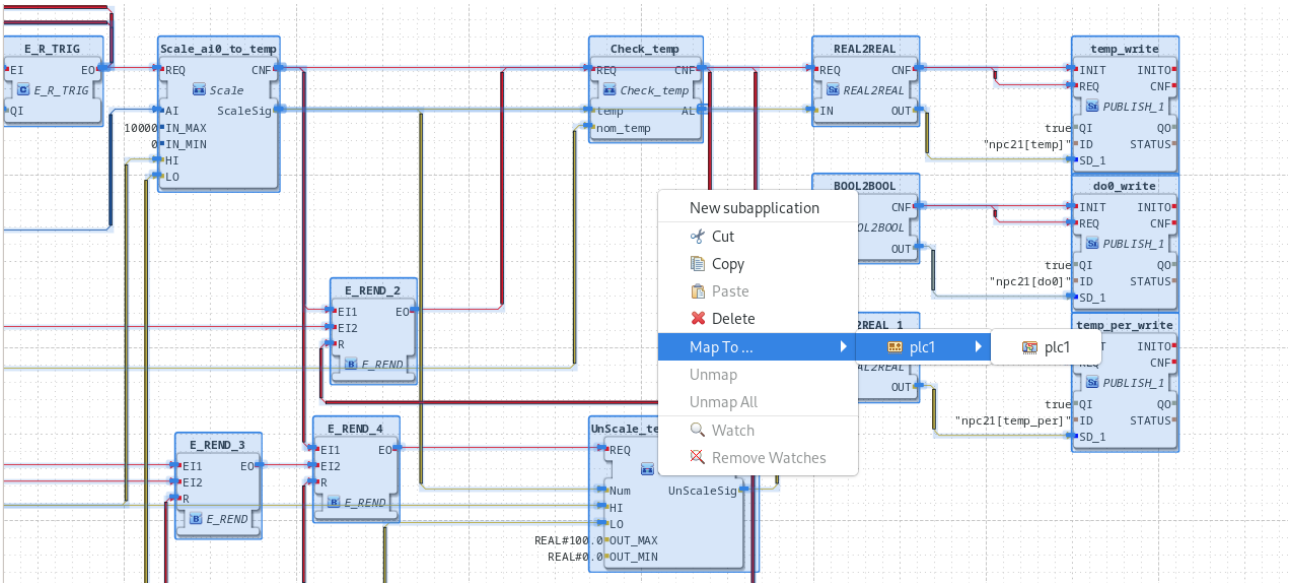

Рисунок 8.44 – Добавление программы

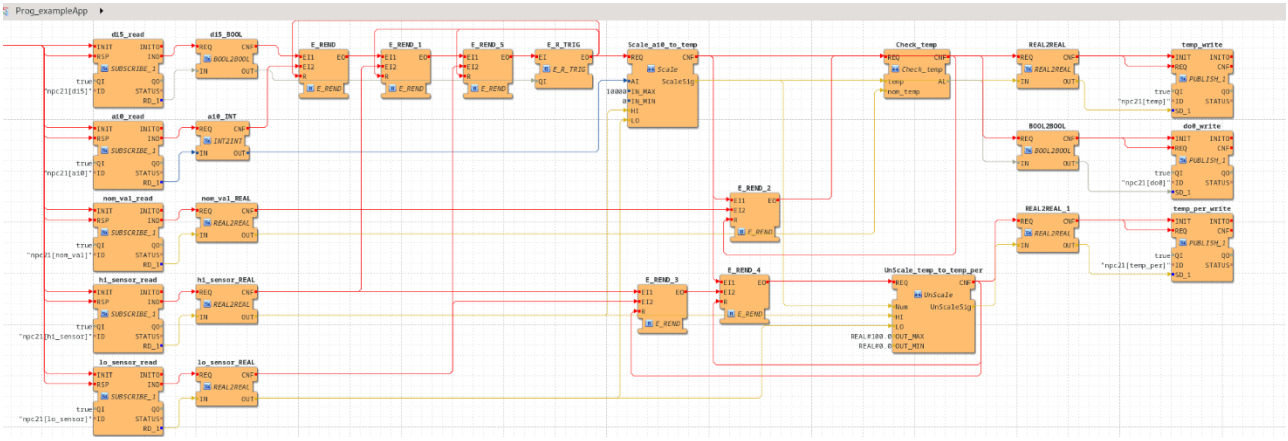

Рисунок 8.45 – Результат добавления программы

Теперь, необходимо зайти в наше ядро *Prog_example->Prog_example->System Configuration/plc1->plc1*. Результат представлен на рисунке 8.46. Как видно, вся программа скопировалось в ядро устройства и теперь необходимо соединить выхода блока *START* с блоком *E_CYCLE*. В данном случае, *COLD* и *WARM* можно соединить с *START*, *STOP* с *STOP*.

Также стоит помнить, что даже при перезагрузке ПЛК в режиме теплого перезапуска он все равно перезапустит Forte среду так, что она интерпретирует это как холодный старт. Итоговая схема представлена на рисунке 8.47.

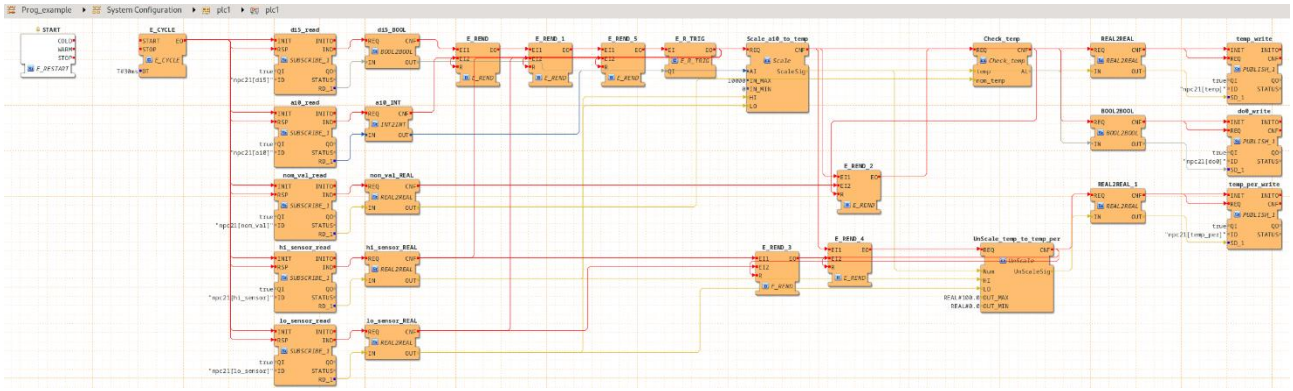

Рисунок 8.46 – Окно программы после добавления в ядро *plc1*

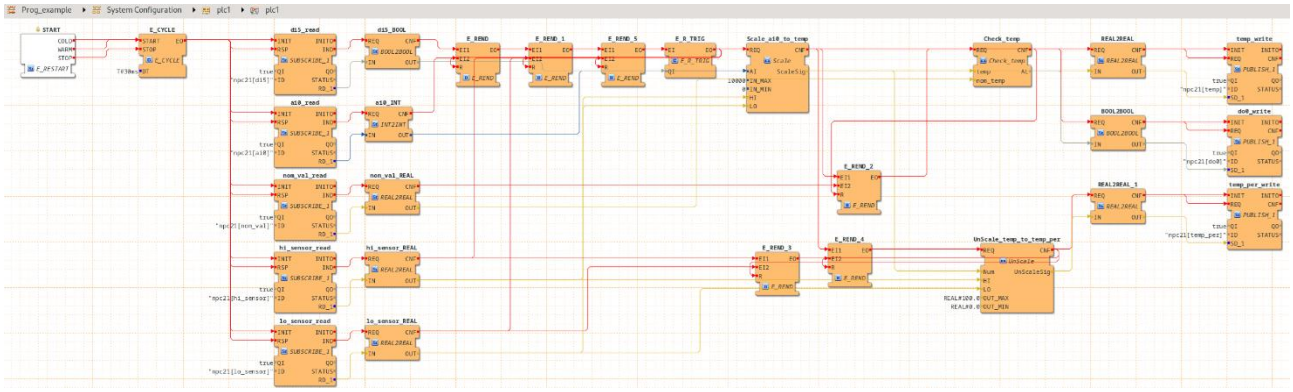

Рисунок 8.47 – Готовая программа к загрузке

Для загрузки программы непосредственно в ПЛК и просмотра программы в режиме онлайн необходимо воспользоваться командой *Deploy*. Она доступна либо для каждого отдельного ядра, либо для каждого устройства, либо для всех устройств и всех их ядер, в зависимости от выбора. Если выбрать ядро (в данной случае *plc1*) и нажать правой кнопкой мыши – появится окно, где будет доступна команда загрузки только в это ядро. Если на устройство – эта команда выполнит загрузку во все ядра в устройстве и т.д. В рамках этого проекта будет загружаться только одно ядро. Шаги загрузки показаны на рисунке 8.48.

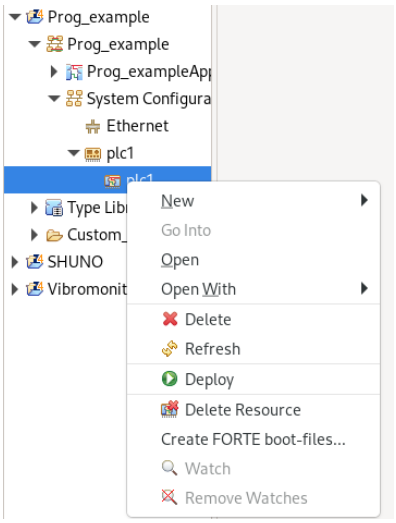

Рисунок 8.48 – Процесс загрузки

Процесс загрузки представлен на рисунке 8.49. Если во время загрузки возникнут ошибки, программа сообщит об этом и ошибку можно будет увидеть в *Deployment Console* ниже окна редактирования программы. Для выхода в режим онлайн необходимо выделить нужные блока, нажать правую кнопку мыши и выбрать *Watch* (смотреть рисунок 8.50 и 8.51). Если во время просмотра везде (особенно на событиях) есть надпись *N/A*, то это говорит о том, что программа, возможно, начала исполнение, но 4diac не получил ответа о состоянии *Deploy* ПЛК. Для решения этой проблемы нужно повторить процедуру загрузки ПО в режиме онлайн. Иногда, это необходимо сделать 4-5 раз.

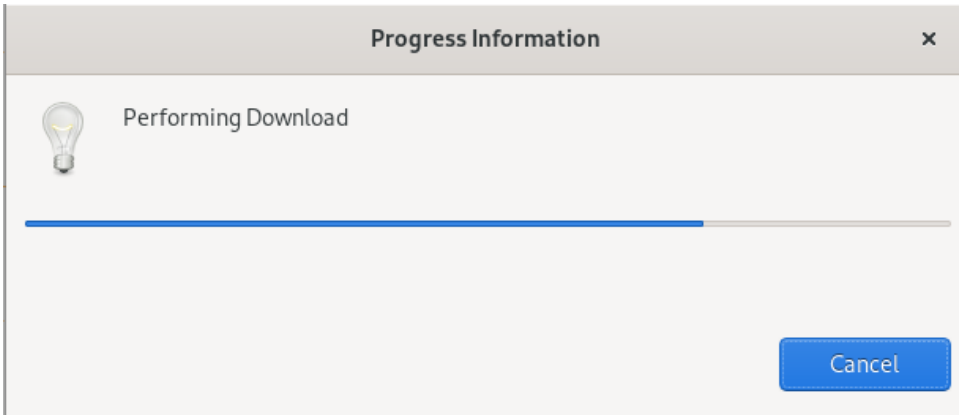

Рисунок 8.49 – Процесс загрузки

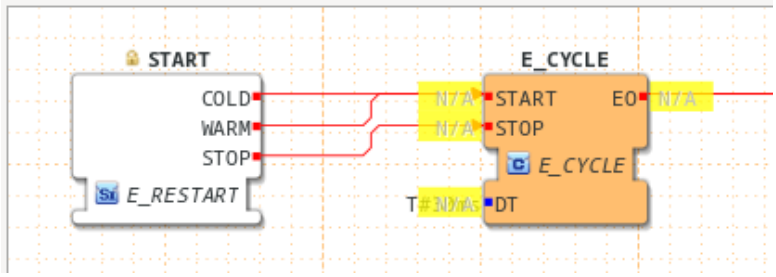

Рисунок 8.50 – Ошибка *N/A*

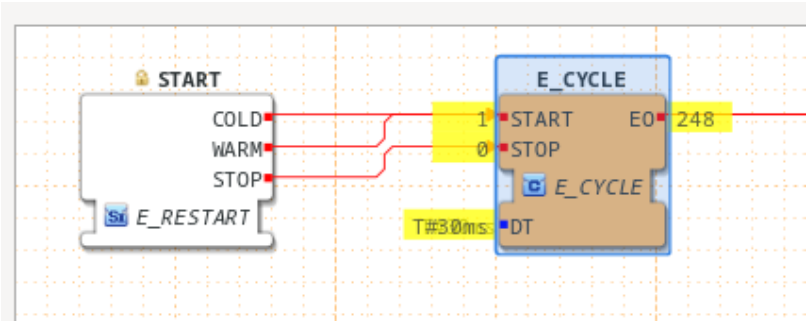

Рисунок 8.51 – Нормальная работа программы

Если выход *EO* блока *E_CYCLE* имеет постоянно увеличивающиеся цифры (события), то это свидетельствует о работоспособности программы. Также, в качестве рекомендации необходимо сказать, что просмотр через ядро не совсем удобен. Если на блоке типа *Subapplication Type* попытаться включить *Watch*, то вылезет ошибка, представленная на рисунке 8.52. Для ее исключения лучше всего мониторить значения через *App (Prog_examopleAPP)*.

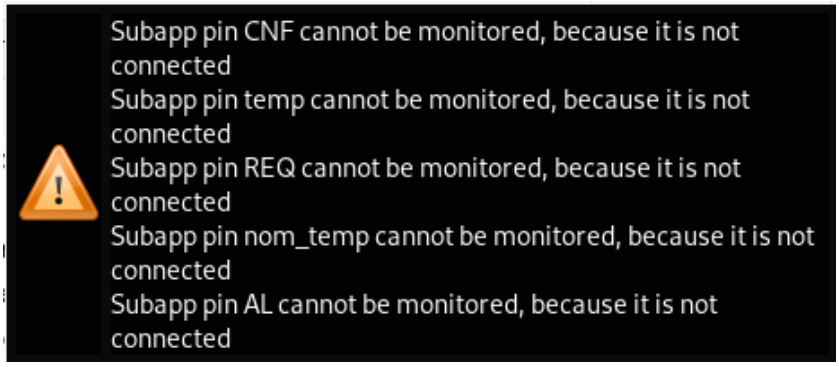

Рисунок 8.52 – Ошибка мониторинга

Пример мониторинга представлен на рисунке 8.53.

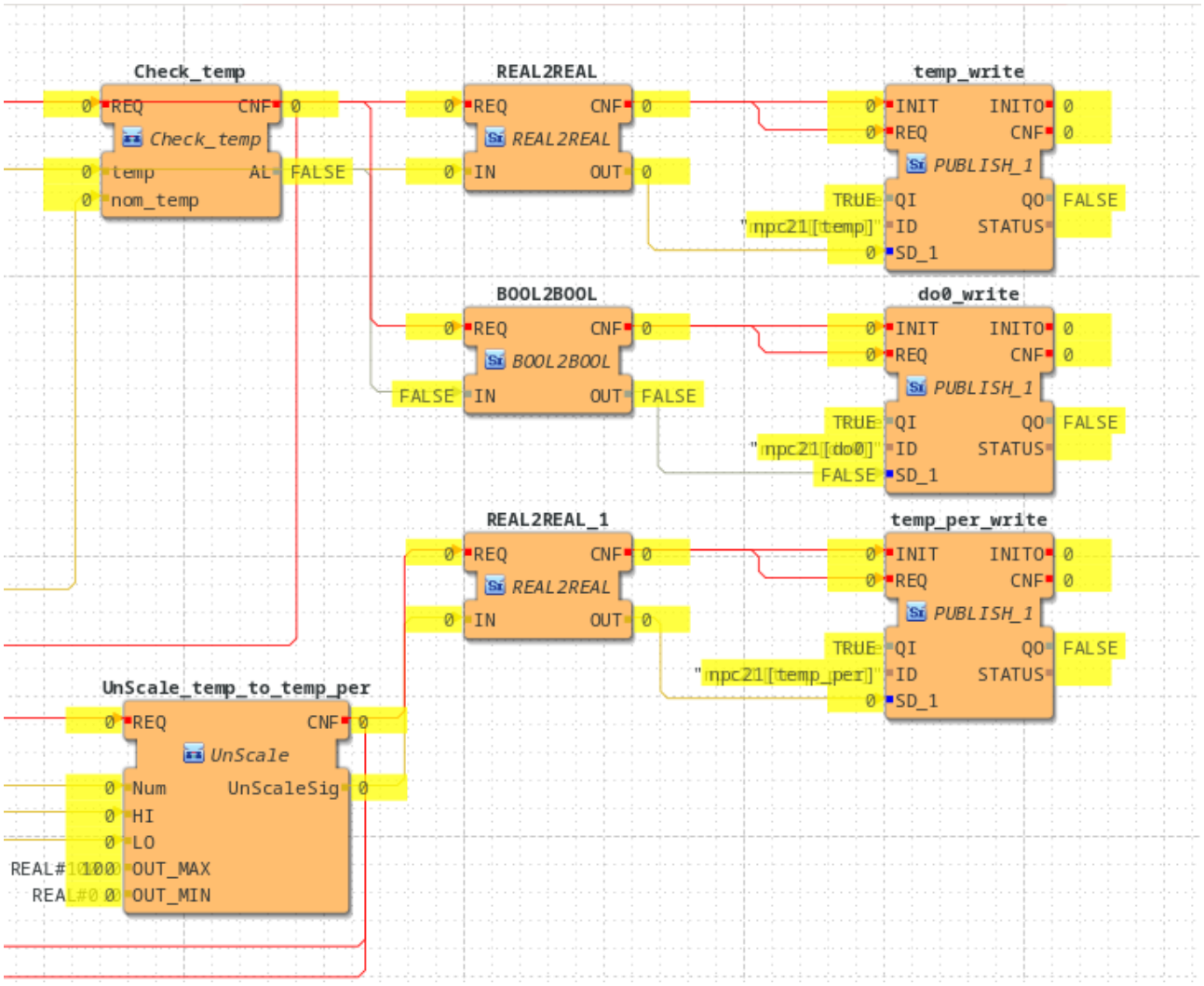

Рисунок 8.53 – Пример мониторинга

Как видно, при отсутствии нажатии кнопки *di5*, вся работа системы остановлена и ждет появления условия (данный алгоритм можно реализовать и по-другому, но в рамках демонстрации событийного программирования была задумана реализация именно через события). При нажатии кнопки результат имеет вид, представленный на рисунке 8.54.

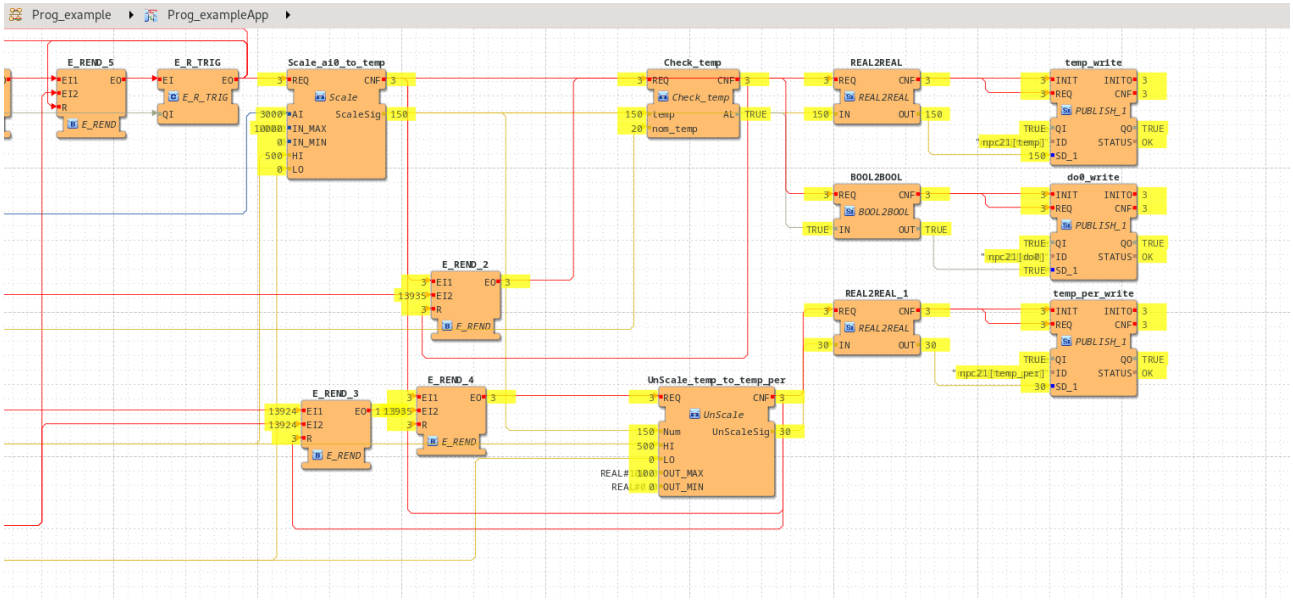

Рисунок 8.54 – Результат

Как видно из данного рисунка, система отработала верно и выполнила все условия задачи. В очередной раз, хочется напомнить, что реализуемая логика не соответствует системе реального времени, а значит, пока нет нажатия *di5*, система не будет реагировать на любые изменения, что демонстрирует принцип событийного программирования.

8.11 Генерация файла *fboot*

После отладки ПО необходимо записать исполняемый файл в постоянную память ПЛК, так как команда *Deploy* записывает проект в оперативную память контроллера. Для этого существуют специальный формат, называемый *fboot*-файлом. Для его создания необходимо выбрать устройство в дереве проекта, нажать правой кнопкой мыши и выбрать *Create FORTE boot-files…* (смотреть рисунок 8.55). Далее появится окно (смотреть рисунок 8.56), где нужно выбрать все библиотеки, которые нужно загрузить, и устройство с ядром, которое необходимо добавить в генерацию исполняемой программы *Forte*. Также следует выбрать путь создания файла. Желательно выбирать путь, имеющий только английский шрифт, где отсутствует специальные символы или пробелы, иначе при загрузке придется текст пути брать в кавычки. После нажатия кнопки *Finish*, будет сгенерирован файл, который имеет расширение *.fboot*. При желании имя файла можно будет изменить. Далее, необходимо загрузить в ПЛК командой *atcmd* с ключом *fboot* (для подробной информации смотреть раздел 3.3.8). Для данного проекта команда будет иметь вид *atcmd plc1 –fboot /home/anonymous/4diacIDE-workspace/Prog_example/Prog_example_plc1.fboot*. После загрузки файла необходимо перезагрузить ПЛК либо вручную, либо через команду (для подробной информации смотреть раздел 3.3.14).

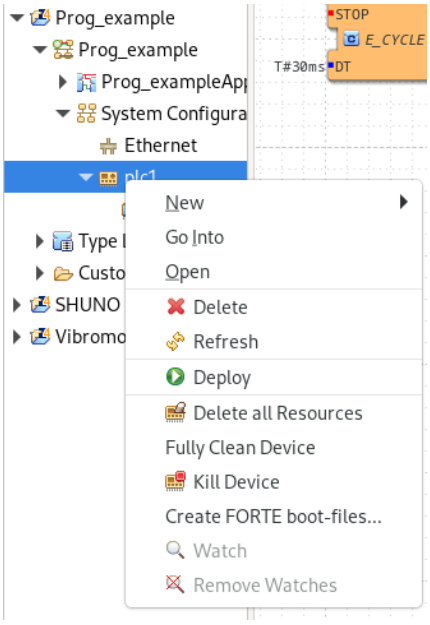

Рисунок 8.55 – Выбор в меню

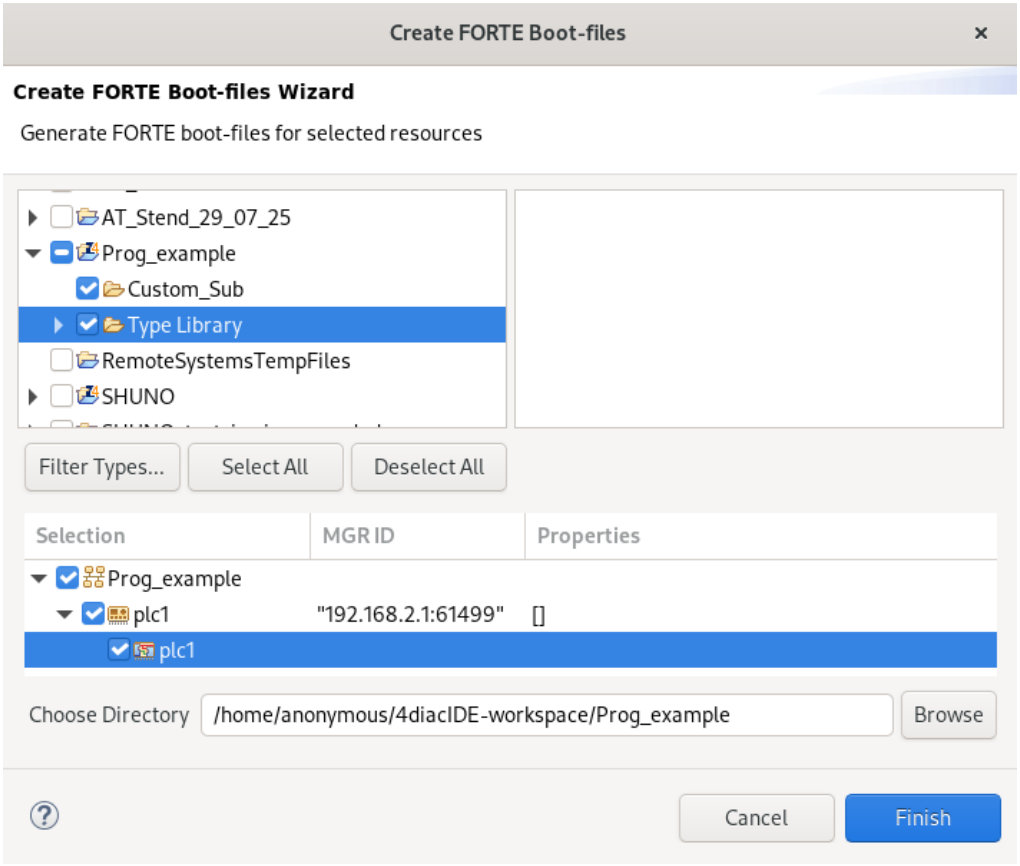

Рисунок 8.56 – Создание файла

На этом разработка ПО для ПЛК посредством программной среды 4diac IDE считается завершенной.

В данном разделе были охвачены основные этапы и принципы событийного программирования, при этом при возникновении вопросов, можно обратиться либо к интегратору, либо напрямую к производителю.

8.12 Разработка программного кода на языке C++

Данная среда программирования позволяет создавать функциональные блоки с использованием языка *C++*. При этом стоит отметить, что для работоспособности кода его необходимо скомпилировать и собрать с ним исходную среду *Forte*, в противном случае, при загрузке возникнет ошибка типа *No Such Object Supported* или *No Such Object Exists*. В настоящее время ПЛК поддерживает разработку, но не поддерживает перекомпиляцию среды *forte*, так как данная функция находится в тестовом режиме и проверяется на устойчивость и стабильность использования С-кодов для промышленного применения. Но при желании воспользоваться данной функцией, советуем обратиться либо к интегратору, либо напрямую к производителю.

Список использованных источников

1. ГОСТ 19.504-79. ЕСПД. Руководство программиста. Требования к содержанию и оформлению. — М. : Стандартинформ, 1979.

2. The Official YAML Web Site. — 2004. — Access mode: https://yaml.org.

3. MATIEC - IEC 61131-3 compiler. — 2012. — Access mode: https://github.com/beremiz/matiec.

4. ESP-IDF Programming Guide. — https://docs.espressif.com/projects/esp-idf/en/latest/. — [Интернет-ресурс, дата доступа 29.10.2023].

5. Eclipse 4diac(TM) - Open Source for Industrial Automation Systems. — 2004. — Access mode: https://eclipse.dev/4diac/.

6. ГОСТ Р МЭК 61131-3-2016. Контроллеры программируемые. Часть 3. Языки программирования. — 2004.

7. MODBUS Application Protocol Specification V1.1b. https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf. — 2006. — [Интернет-ресурс, дата доступа 29.10.2023].

8. ГОСТ Р МЭК 60870-5-104-2004. Устройства и системы телемеханики. Часть 5. Протоколы передачи. Раздел 104. Доступ к сети для ГОСТ Р МЭК 870-5-101 с использованием стандартных транспортных профилей. — 2004.

9. Репозиторий IT-P Engineering. — 2026. — Access mode: https://github.com/ITPEngineering/Ai-Talap

10. Regex(7)—Linux manual page. — 2024. — Access mode: https://www.man7.org/linux/man-pages/man7/regex.7.html.

11. ГОСТ Р МЭК 870-5-5. Устройства и системы телемеханики. Часть 5. Протоколы передачи. Раздел 5. Основные прикладные функции. — 1996.

12. ГОСТ Р МЭК 60870-5-101-2006. Устройства и системы телемеханики. Часть 5. Протоколы передачи. Раздел 101. Обобщающий стандарт по основным функциям телемеханики. — 2006.