Beremiz_sofi

Ruslan Kid

июн. 14, 2023

Содержание:

Среда разработки «Beremiz» предназначена для создания и отладки прикладных программ на языках стандарта IEC 61131-3. В качестве ПЛК в данных уроках будет использоваться BRIC и его модули расширения – разработка ООО «СНЭМА-СЕРВИС». В качестве языков описания алгоритмов и логики работы данных программ будут выступать Structured Text (далее ST) и Function Block Diagram (далее FBD).

Глава 1

Знакомство со средой Beremiz

Примечание: Более подробную информацию о создании, настройке и загрузке пользовательской программы в ПЛК BRIC можно узнать по ссылке.

1.1 Инструкции по установке и обновлению Beremiz

Инсталлятор в формате «.exe» можно скачать кликнув по 

При переходе по ссылке открывается страница загрузки файла «Beremiz_BRIC_0_12_0_4_Setup.exe».

После скачивания необходимо запустить файл, дважды кликнув левой клавишей мыши. Открывается окно установки, в котором предлагается выбрать директорию.

После выбора папки необходимо нажать кнопку «OK», с момента которого начинается процесс распаковки файлов.

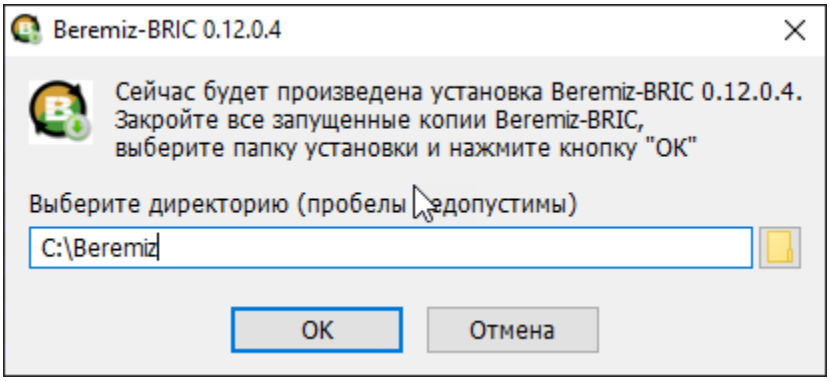
При успешной установке Beremiz выводится окно уведомления.

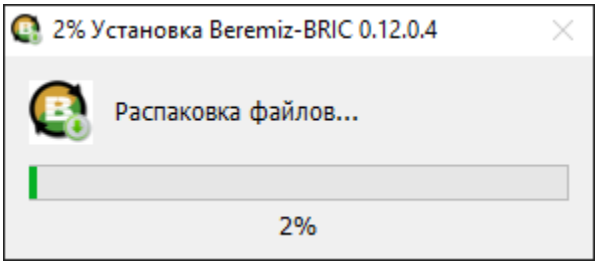
В результате установки в выбранной директории появится папка «Beremiz», в котором содержатся файлы и папки, представленные на рисунке ниже. Также появится ярлык на рабочем столе.

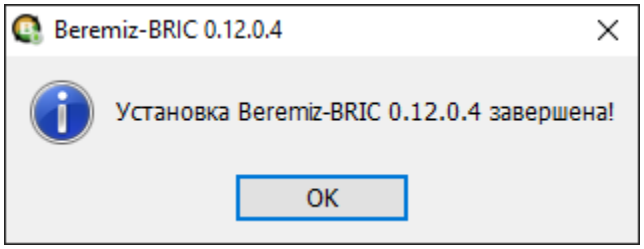
Запуск Beremiz осуществляется двойным нажатием левой клавишей мыши по файлу «Beremiz.exe», либо по ярлыку на рабочем столе.

Для обновления необходимо кликнуть по Help -> Updater и нажать на кнопку Update. В консоли выдается сообщении об обновлении разделов. Для применения обновлений необходимо перезапустить программу.

Beremiz_BRIC_0_12_0_4_Setup.exe



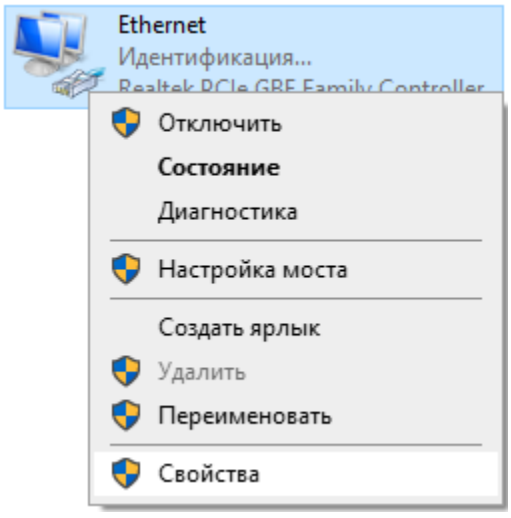




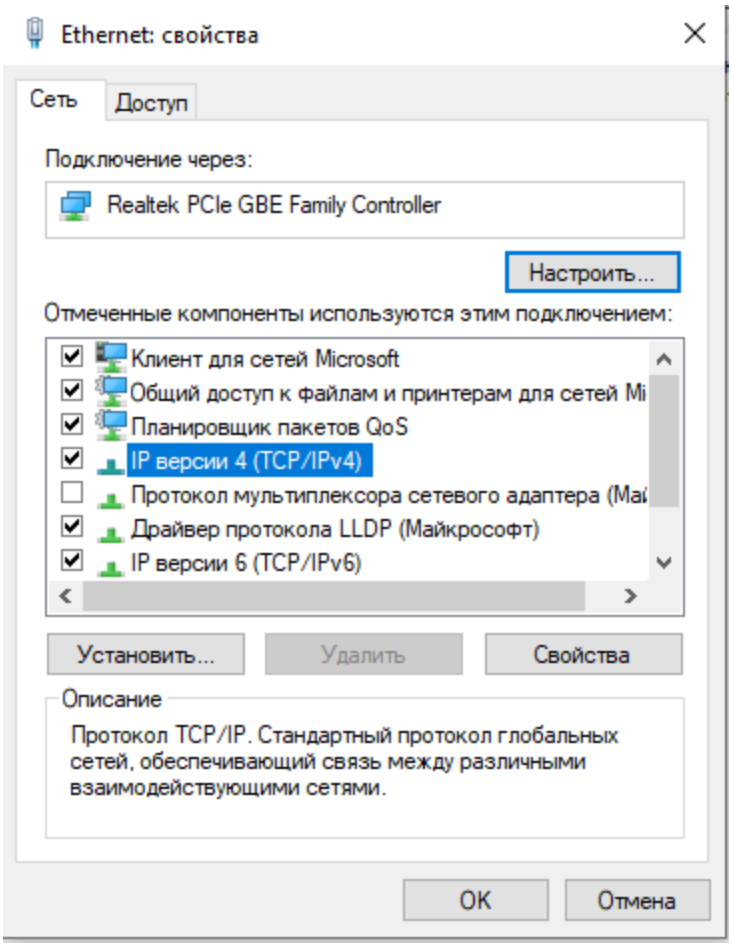
Имя	Дата изменения	Тип	Размер
beremiz	28.03.2022 9:55	Папка с файлами	
bric_examples	28.03.2022 9:55	Папка с файлами	
build	28.03.2022 9:54	Папка с файлами	
cmake-3-13-1-win32-x86	28.03.2022 9:54	Папка с файлами	
gcc_6_3_1	28.03.2022 9:54	Папка с файлами	
matiec	28.03.2022 9:55	Папка с файлами	
mingw	28.03.2022 9:55	Папка с файлами	
PortableGit	28.03.2022 9:55	Папка с файлами	
python2	28.03.2022 9:55	Папка с файлами	
python3	28.03.2022 9:56	Папка с файлами	
sofi	28.03.2022 9:54	Папка с файлами	
Uninstall	28.03.2022 9:55	Папка с файлами	
Beremiz.exe	17.02.2022 14:50	Приложение	389 КБ
license.txt	24.01.2017 5:12	Файл "TXT"	18 КБ

1.2 Пошаговая инструкция настройки параметров сетевой карты ПК для подключения по LAN

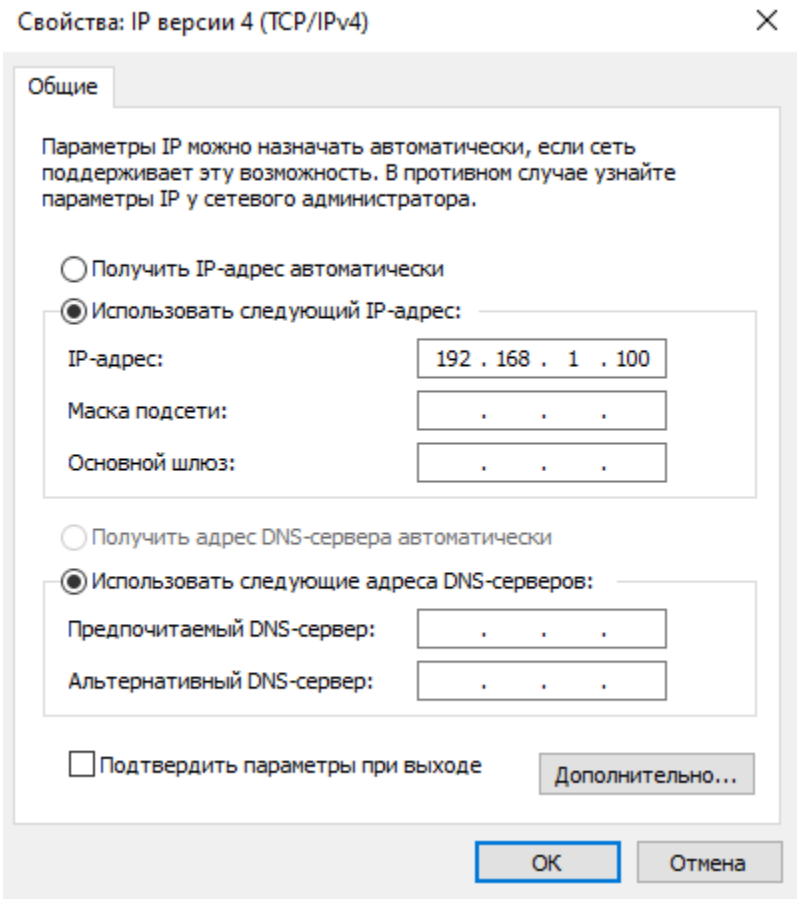
- Подключаем ПК к контроллеру с помощью LAN-кабеля.
- Заходим в **Панель управления** -> **Сеть и Интернет** -> **Сетевые подключения**.
- Выбираем сеть Ethernet, нажимаем правой кнопкой мыши в свойства адаптера.



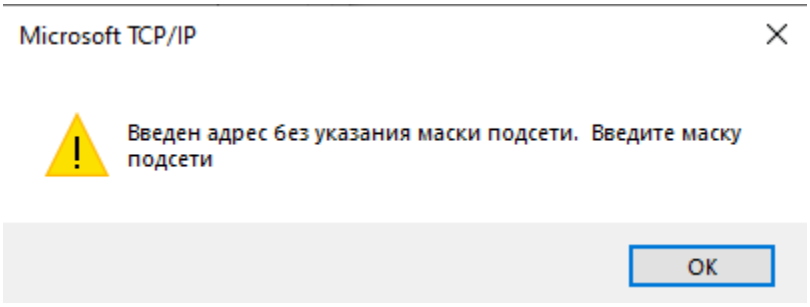
- Необходимо настроить протокол TCP/IPv4, для этого дважды нажимаем левой кнопки мыши по IP версии 4.


Ethernet: свойства
Сеть
Доступ
Подключение через:
Realtek PCIe GBE Family Controller
Настроить...
Отмеченные компоненты используются этим подключением:
Клиент для сетей Microsoft
Общий доступ к файлам и принтерам для сетей Mi
Планировщик пакетов QoS
IP версии 4 (TCP/IPv4)
Протокол мультиплексора сетевого адаптера (Ma
Драйвер протокола LLDP (Майкрософт)
IP версии 6 (TCP/IPv6)
Установить...
Удалить
Свойства
Описание
Протокол TCP/IP. Стандартный протокол глобальных сетей, обеспечивающий связь между различными взаимодействующими сетями.
OK
Отмена

- По умолчанию при подключении через интерфейс Ethernet IP-адрес: 192.168.1.232, поэтому используем IP-адрес 192.168.1.XXX чтобы оказаться в одной подсети с контроллером. Например:



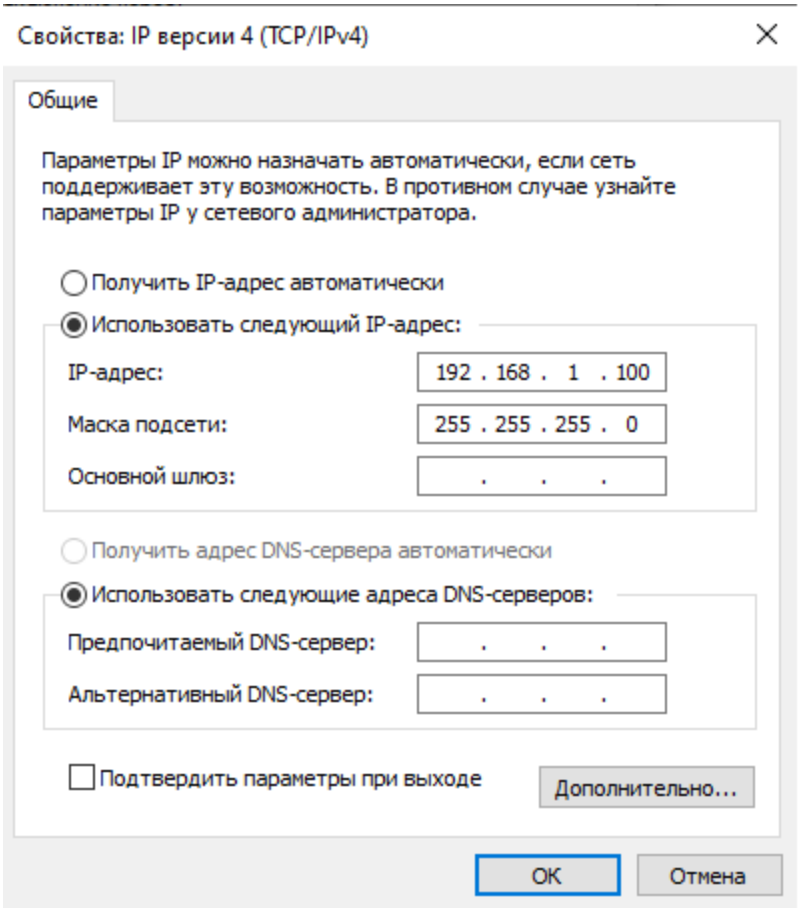
- Нажимаем кнопку «OK». Выводится сообщение о введении адреса без указания маски подсети.



Нажимаем кнопку «OK». По умолчанию выставляется маска подсети: «255.255.255.0»

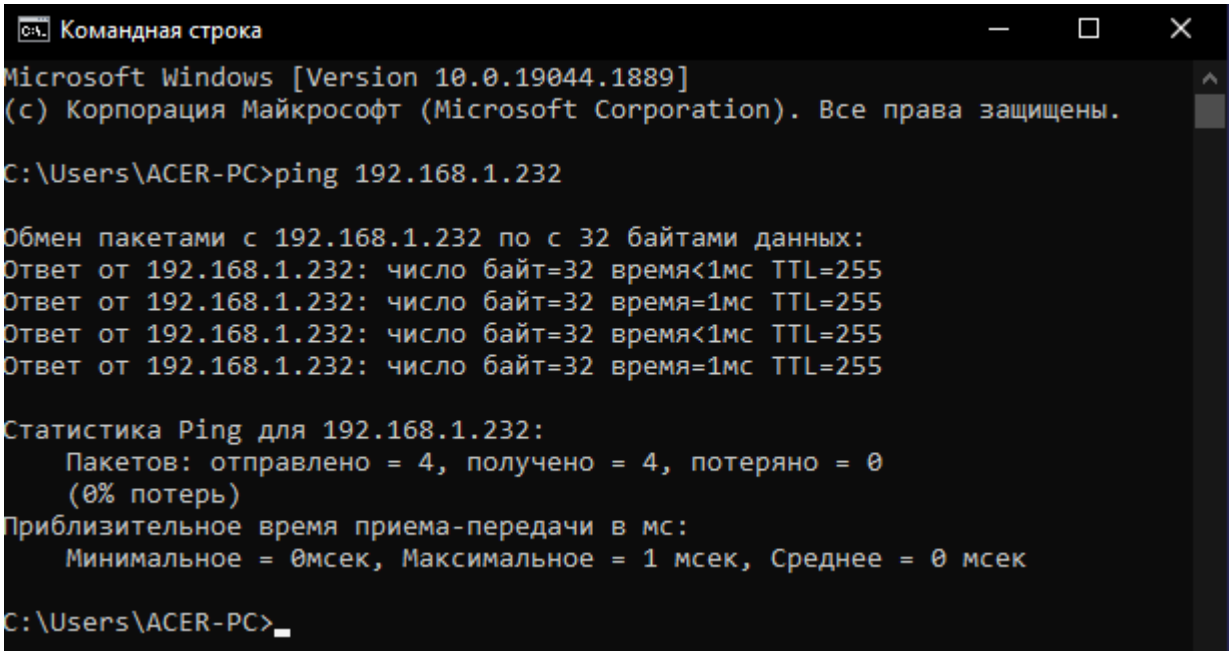
- Для завершения настройки параметров нажимаем кнопку «OK».

- Для проверки связи ПК с контроллером заходим в **Командную строку** и прописываем:


Свойства: IP версии 4 (TCP/IPv4)
Общие
Параметры IP можно назначать автоматически, если сеть поддерживает эту возможность. В противном случае узнайте параметры IP у сетевого администратора.
Получить IP-адрес автоматически
Использовать следующий IP-адрес:
IP-адрес:
192 . 168 . 1 . 100
Маска подсети:
255 . 255 . 255 . 0
Основной шлюз:
Получить адрес DNS-сервера автоматически
Использовать следующие адреса DNS-серверов:
Предпочитаемый DNS-сервер:
Альтернативный DNS-сервер:
Подтвердить параметры при выходе
Дополнительно...
OK
Отмена

```
ping 192.168.1.232
```

Нажимаем клавишу «Enter». Выводится сообщение об успешном обмене пакетами данных.

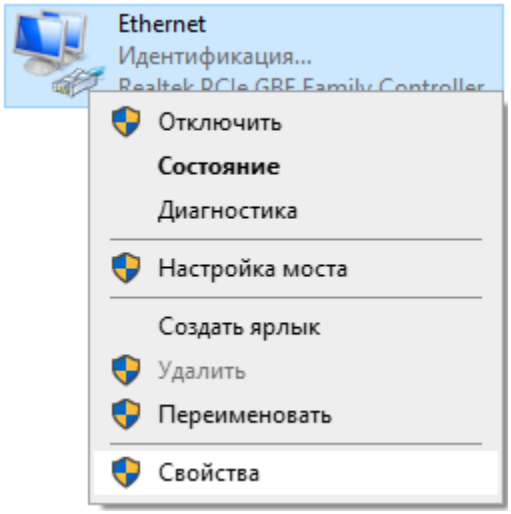


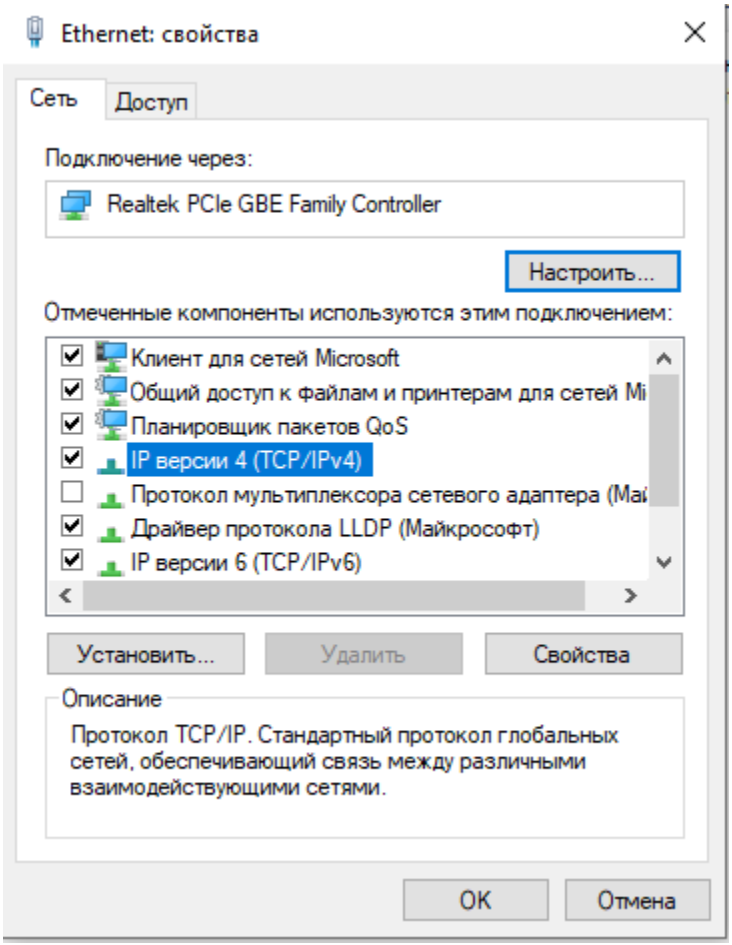
1.3 Пошаговая инструкция настройки параметров сетевой карты ПК для подключения по USB

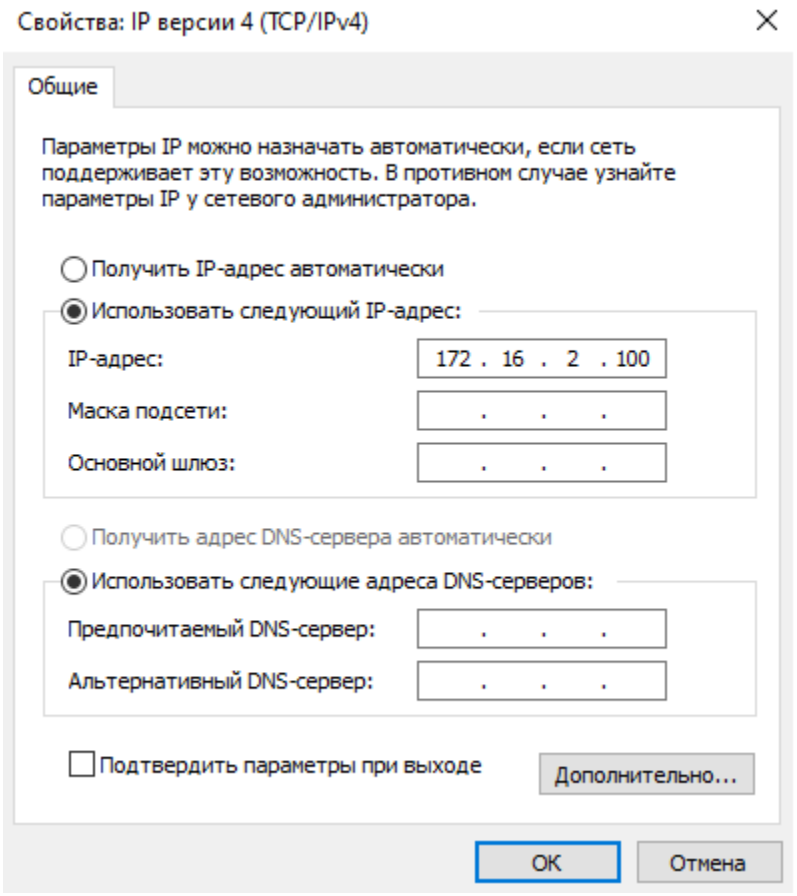
- Подключаем ПК к контроллеру с помощью USB-кабеля.
- Заходим в **Панель управления** -> **Сеть и Интернет** -> **Сетевые подключения**.
- Выбираем сеть Ethernet (удостоверившись, что эта сеть подключения по USB), нажимаем правой кнопкой мыши в свойства адаптера.
- Необходимо настроить протокол TCP/IPv4, для этого дважды нажимаем левой кнопки мыши по IP версии 4.
- По умолчанию при подключении через интерфейс Ethernet IP-адрес: 172.16.2.232, поэтому используем IP-адрес 172.16.2.XXX чтобы оказаться в одной подсети с контроллером. Например:
- Нажимаем кнопку «OK». Выводится сообщение о введении адреса без указания маски подсети.

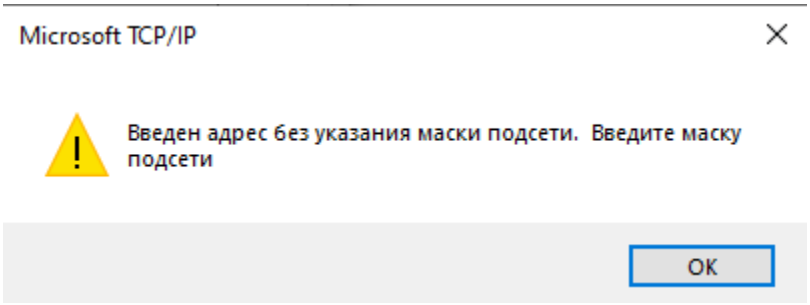
Нажимаем кнопку «OK». По умолчанию выставляется маска подсети: «255.255.255.0»

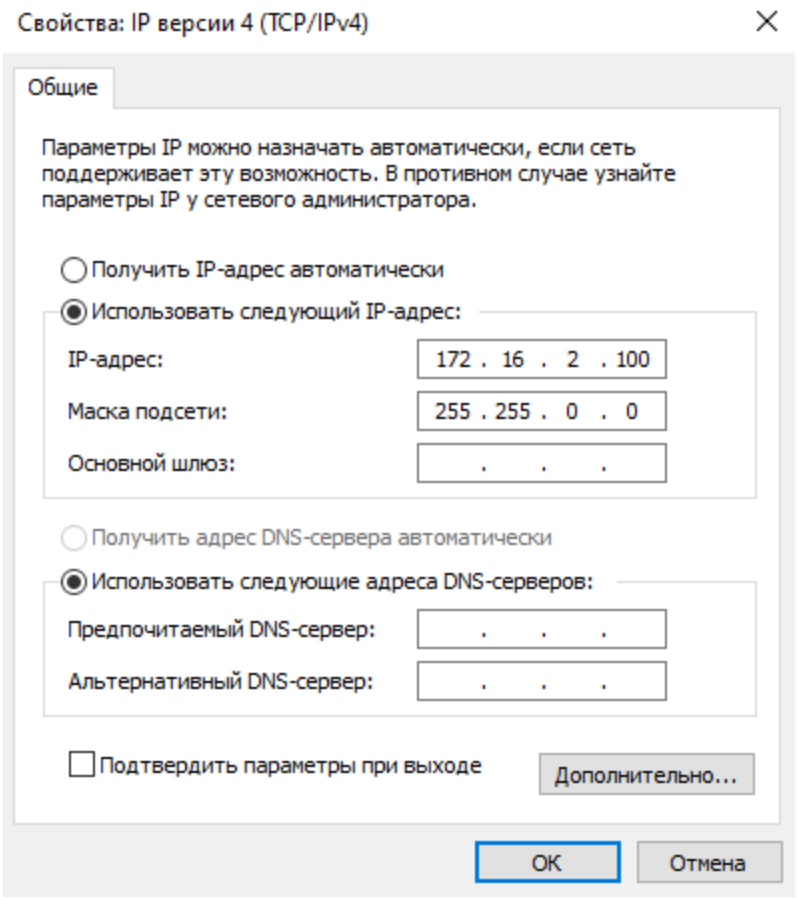
- Для завершения настройки параметров нажимаем кнопку «OK».


Ethernet
Идентификация...
Realtek PCIe GBE Family Controller
Отключить
Состояние
Диагностика
Настройка моста
Создать ярлык
Удалить
Переименовать
Свойства


Ethernet: свойства
Сеть
Доступ
Подключение через:
Realtek PCIe GBE Family Controller
Настроить...
Отмеченные компоненты используются этим подключением:
Клиент для сетей Microsoft
Общий доступ к файлам и принтерам для сетей Mi
Планировщик пакетов QoS
IP версии 4 (TCP/IPv4)
Протокол мультиплексора сетевого адаптера (Ма
Драйвер протокола LLDP (Майкрософт)
IP версии 6 (TCP/IPv6)
Установить...
Удалить
Свойства
Описание
Протокол TCP/IP. Стандартный протокол глобальных сетей, обеспечивающий связь между различными взаимодействующими сетями.
OK
Отмена


Свойства: IP версии 4 (TCP/IPv4)
Общие
Параметры IP можно назначать автоматически, если сеть поддерживает эту возможность. В противном случае узнайте параметры IP у сетевого администратора.
Получить IP-адрес автоматически
Использовать следующий IP-адрес:
IP-адрес:
172 . 16 . 2 . 100
Маска подсети:
Основной шлюз:
Получить адрес DNS-сервера автоматически
Использовать следующие адреса DNS-серверов:
Предпочитаемый DNS-сервер:
Альтернативный DNS-сервер:
Подтвердить параметры при выходе
Дополнительно...
OK
Отмена

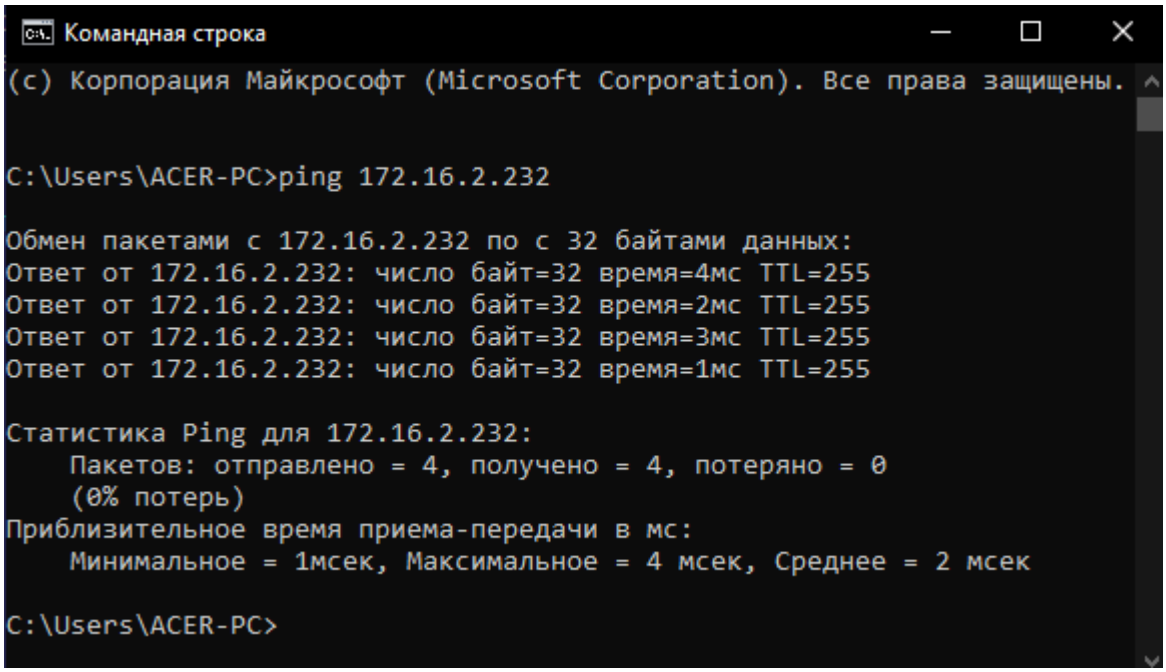

Microsoft TCP/IP
Введен адрес без указания маски подсети. Введите маску подсети
OK


Свойства: IP версии 4 (TCP/IPv4)
Общие
Параметры IP можно назначать автоматически, если сеть поддерживает эту возможность. В противном случае узнайте параметры IP у сетевого администратора.
Получить IP-адрес автоматически
Использовать следующий IP-адрес:
IP-адрес:
172 . 16 . 2 . 100
Маска подсети:
255 . 255 . 0 . 0
Основной шлюз:
Получить адрес DNS-сервера автоматически
Использовать следующие адреса DNS-серверов:
Предпочитаемый DNS-сервер:
Альтернативный DNS-сервер:
Подтвердить параметры при выходе
Дополнительно...
OK
Отмена

- Для проверки связи ПК с контроллером заходим в **Командную строку** и прописываем:

```
ping 172.16.2.232
```

Нажимаем клавишу «Enter». Выводится сообщение об успешном обмене пакетами данных.



1.4 Настройка проекта

Проект в Beremiz представляет собой именованную папку, в которой лежат исходные файлы. Она должна быть обязательно пустой и не защищенной от записи. Если там уже есть файлы, будет выдана соответствующая ошибка. В созданной папке будут сохранены следующие файлы и папки:

- «beremiz.xml» – в данном XML файле сохраняются настройки специфичные для среды разработки Beremiz относительно проекта;
- «plc.xml» – в данном XML файле сохраняется полное описание проекта: всех программ, ресурсов, пользовательских типов данных, данных о проекте, настроек редакторов графических языков IEC 61131-3;
- папка «build», которая хранит генерируемый ST и C код, а также получаемый исполняемый бинарный файл пользовательской программы.

Внимание: Название проекта не должно содержать пробел и недопустимые символы

1.5 Создание нового проекта

Новый проект создаётся с помощью главного меню «File» – «New», либо с помощью кнопки «New» на панели управления.

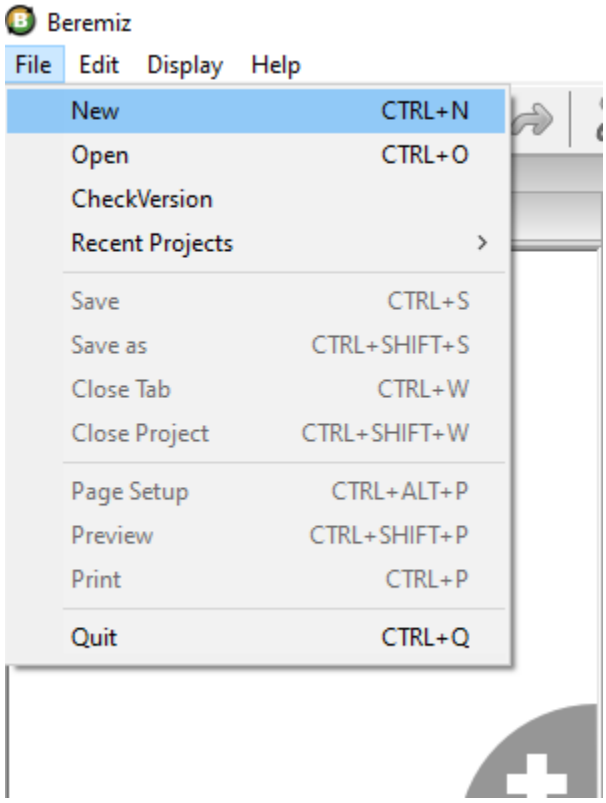


Рис. 1: *Создание нового проекта с помощью главного меню*

Далее появится диалог, в котором необходимо выбрать папку, где будет храниться данный проект.

В появившемся диалоге вам будет предложено настроить основной программный модуль проекта. В данном диалоге три поля:

- «POU Name»;
- «POU Type»;
- «Language».

POU Name - имя программы, присвоенное по умолчанию, может быть заменено на любое имя, соответствующее назначению данной программы.

POU Type – «Program», в дальнейшем в проект можно добавить дополнительные программы, функции и функциональные блоки.

В поле Language необходимо выбрать из списка один из языков стандарта IEC 61131-3 (IL, ST, LD, FBD, SFC), на котором будут реализованы алгоритмы и логика работы данной добавляемой программы.

При нажатии кнопки «OK» в проект будет добавлен основной программный модуль с выбранными параметрами, ресурс проекта будет сконфигурирован по умолчанию: добавлена одна задача циклического выполнения с интервалом 20 мс, и один экземпляр основной программы. При нажатии кнопки «Cancel» будет создан пустой проект без каких-либо настроек.

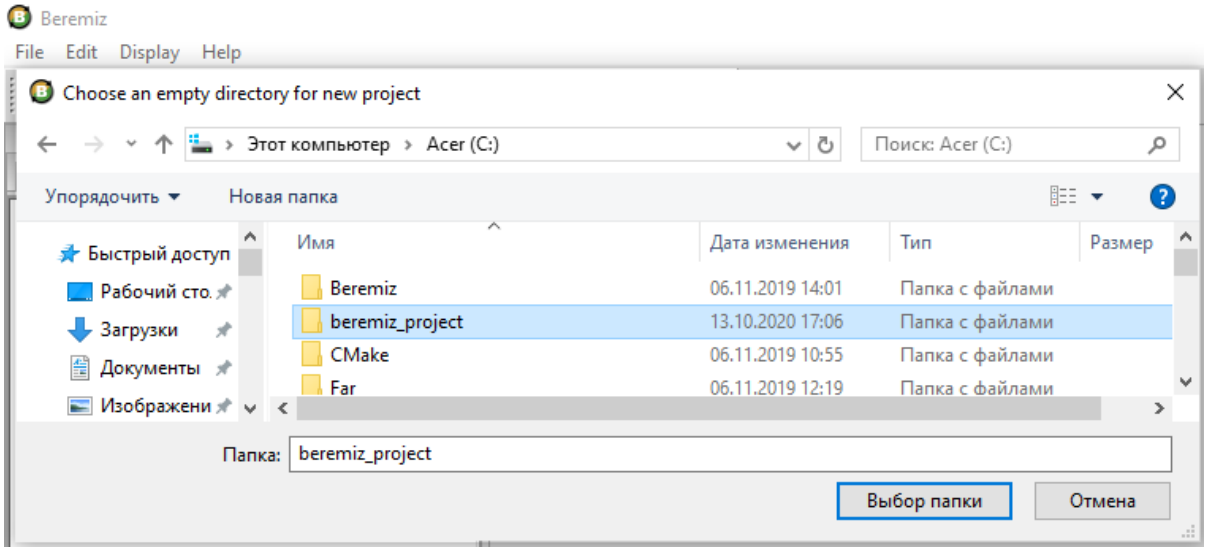


Рис. 2: *Диалог выбора папки для нового проекта*

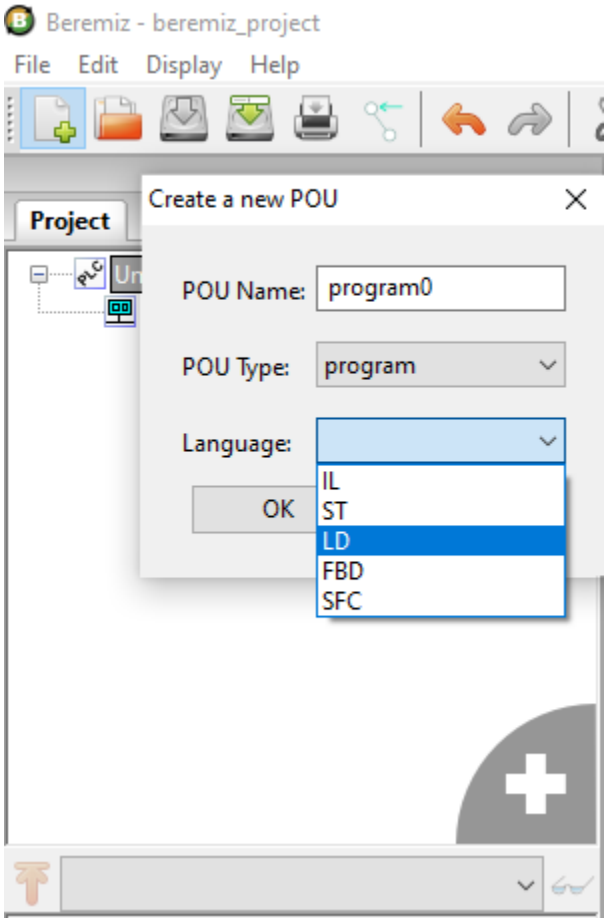
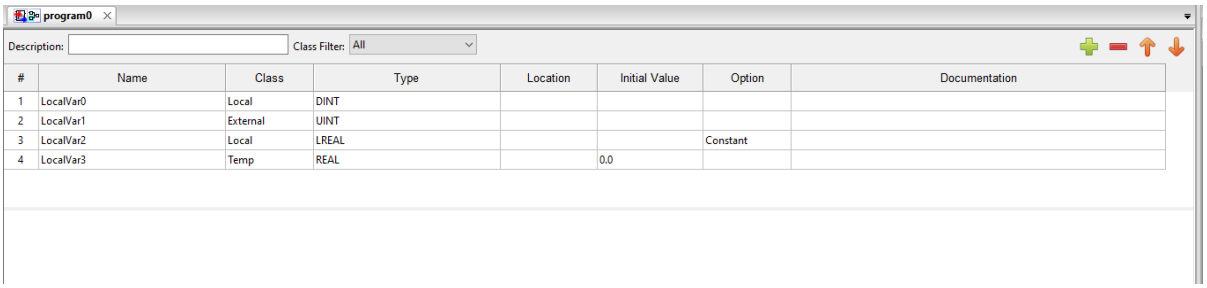


Рис. 3: *Диалог добавления основной программы*

Конфигурационные переменные проекта

Конфигурационные переменные позволяют программным модулям типа «Program» и «Function block» использовать общие переменные, которые будут определены в глобальной области видимости проекта.

В панели переменных и констант добавим стандартные переменные «LocalVar0» типа DINT, с помощью кнопки «Добавить переменную». Таким же образом добавим остальные переменные. На рисунке ниже представлен результат объявления конфигурационных переменных.



#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	LocalVar0	Local	DINT				
2	LocalVar1	External	UINT				
3	LocalVar2	Local	LREAL			Constant	
4	LocalVar3	Temp	REAL		0.0		

Рис. 4: *Объявление конфигурационных переменных*

Для того чтобы к данной конфигурационной переменной можно было обращаться из программных модулей типа «Program» или «Function block» необходимо в их панели редактирования в панели переменных и констант создать переменную с таким же именем, как и ранее объявленная глобальная, и установить её класс «External» (Внешний).

1.6 Настройки сборки проекта и соединения с ПЛК BRIC

Для использования написанной пользовательской программы необходимо её собрать (скомпилировать и скомпоновать), т.е. получить исполняемый файл и передать на ПЛК BRIC. В связи с этим основными настройками являются: «URI_location» - адрес ПЛК, и «TargetType» - архитектура платформы.

Примечание: IP-адрес при подключении через Ethernet-порт по умолчанию - *192.168.1.232*

Внимание: ПЛК BRIC и модули расширения имеют архитектуру «Sofi», поэтому во вкладке «Config» в разделе TargetType требуется установить платформу «Sofi».

1.7 Языки стандарта МЭК 61131-3

ST (Structured Text) – это текстовый язык высокого уровня общего назначения, по синтаксису схожий с языком Pascal. Удобен для программ, включающих числовой анализ или сложные алгоритмы. Может использоваться в программах, в теле функции или функционального блока, а также для описания действия и перехода внутри элементов SFC. Согласно IEC 61131-3 ключевые слова должны быть введены в символах верхнего регистра. Пробелы и метки табуляции не влияют на синтаксис, они могут использоваться везде.

Выражения в ST выглядят точно также, как и в языке Pascal:

[variable] := [value];

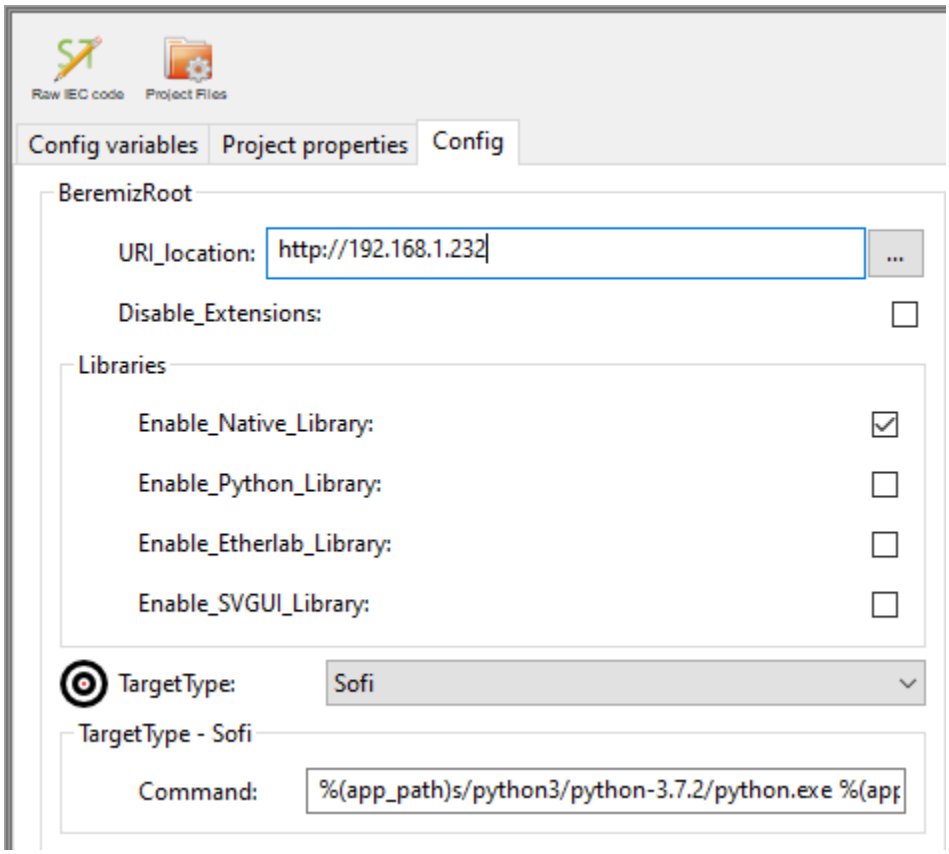


Рис. 5: *Конфигурация проекта*

Порядок их выполнения – справа налево. Выражения состоят из операндов и операторов. Операндом является литерал, переменная, структурированная переменная, компонент структурированной переменной, обращение к функции или прямой адрес.

Конструкции языка IL

IL (Instruction List) представляет собой текстовый язык программирования низкого уровня, который очень похож на Assembler, но к конкретной архитектуре процессора не привязан. Он позволяет описывать функции, функциональные блоки и программы, а также шаги и переходы в языке SFC. Одним из ключевых преимуществ IL является его простота и возможность добиться оптимизированного кода для реализации критических секторов программ. Особенности IL делают его неудобным для описания сложных алгоритмов с большим количеством разветвлений.

Основа языка программирования IL, как и в случае Assembler, это переходы по меткам и аккумулятор. В аккумулятор загружается значения переменной, а дальнейшее выполнение алгоритма представляет собой извлечение значения из аккумулятора и совершение над ним операций. В таблице приведены операторы языка IL.

Таблица 1: Операторы языка IL

Оператор	Описание
LD	Загрузить значение операнда в аккумулятор
LDN	Загрузить обратное значение операнда в аккумулятор
ST	Присвоить значение аккумулятора операнду
STN	Присвоить обратное значение аккумулятора операнду
S	Если значение аккумулятора TRUE, установить логический операнд
R	Если значение аккумулятора FALSE, сбросить логический операнд
AND	«Поразрядное И» аккумулятора и операнда
ANDN	«Поразрядное И» аккумулятора и обратного операнда
OR	«Поразрядное ИЛИ» аккумулятора и операнда
ORN	«Поразрядное ИЛИ» аккумулятора и обратного операнда
XOR	«Поразрядное разделительное ИЛИ» аккумулятора и операнда
XORN	«Поразрядное разделительное ИЛИ» аккумулятора и обратного операнда
ADD	Сложение аккумулятора и операнда, результат записывается в аккумулятор
SUB	Вычитание операнда из аккумулятора, результат записывается в аккумулятор
MUL	Умножение аккумулятора на операнд, результат записывается в аккумулятор
DIV	Деление аккумулятора на операнд, результат записывается в аккумулятор
GT	Значение аккумулятора сравнивается со значением операнда(>(greater than)). Значение (TRUE или FALSE) записывается в аккумулятор
GE	Значение аккумулятора сравнивается со значением операнда (>=greater than or equal). Значение (TRUE или FALSE) записывается в аккумулятор
EQ	Значение аккумулятора сравнивается со значением операнда (=(equal)). Значение (TRUE или FALSE) записывается в аккумулятор
NE	Значение аккумулятора сравнивается со значением операнда <>(not equal). Значение (TRUE или FALSE) записывается в аккумулятор
LE	Значение аккумулятора сравнивается со значением операнда (<=(less than or equal to)). Значение (TRUE или FALSE) записывается в аккумулятор
LT	Значение аккумулятора сравнивается со значением операнда (<(less than)). Значение (TRUE или FALSE) записывается в аккумулятор
JMP	Переход к метке
JMPC	Переход к метке при условии, что значение аккумулятора TRUE
JMPCN	Переход к метке при условии, что значение аккумулятора FALSE

LD (Ladder Diagram) – графический язык, основанный на принципах релейно-контактных схем

(элементами релейно-контактной логики являются: контакты, обмотки реле, вертикальные и горизонтальные перемычки и др.) с возможностью использования большого количества различных функциональных блоков. Достоинствами языка LD являются: представление программы в виде электрического потока (близко специалистам по электротехнике), наличие простых правил, использование только булевых выражений. Схемы, реализованные на данном языке, называются многоступенчатыми. Они представляют собой набор горизонтальных цепей, напоминающих ступеньки лестницы, соединяющих вертикальные шины питания.

Объекты языка программирования LD обеспечивают средства для структурирования программного модуля в некоторое количество контактов, катушек. Эти объекты взаимосвязаны через фактические параметры или связи.

Порядок обработки индивидуальных объектов в LD-секции определяется потоком данных внутри секции. Ступени, подключенные к левой шине питания, обрабатываются сверху вниз (соединение к левой шине питания). Ступени внутри секции, которые не зависят друг от друга, обрабатываются в порядке размещения.

Слева и справа схема на языке LD ограничена вертикальными линиями – шинами питания. Между ними расположены цепи, образованные контактами и катушками реле, по аналогии с обычными электронными цепями. Слева любая цепь начинается набором контактов, которые посылают слева направо состояние «ON» или «OFF», соответствующие логическим значениям TRUE или FALSE. Каждому контакту соответствует логическая переменная (типа BOOL). Если переменная имеет значение TRUE, то состояние передается через контакт. Иначе – правое соединение получает значение выключено («OFF»).

Контакты могут быть соединены параллельно, тогда соединение передает состояние «логическое ИЛИ». Если контакты соединены последовательно, то соединение передаёт «логическое И».

Контакт может быть инвертируемым. Такой контакт обозначается с помощью символа |/| и передает состояние «ON», если значение переменной FALSE.

SFC (Sequential Function Chart) расшифровывается как «Последовательность функциональных диаграмм», и является одним из языков стандарта IEC 61131-3. SFC позволяет легко описывать последовательность протекания процессов в системе.

SFC осуществляет последовательное управление процессом, базируясь на системе условий, передающих управления с одной операции на другую. Язык SFC состоит из конечного числа базовых элементов, которые используются как блоки для построения целостного алгоритма протекания программы.

Язык SFC использует следующие структурные элементы для создания программы: шаг (и начальный шаг), переход, блок действий, прыжок и связи типа дивергенция и конвергенция.

После вызова программного модуля, описанного языком SFC, первым выполняется начальный шаг. Шаг, выполняемый в данный момент, называется активным. Действия, связанные с активным шагом, выполняются один раз в каждом управляющем цикле. Следующий за активным шагом шаг станет активным, только если в переходе между этими шагами условие будет истинно.

В каждом управляющем цикле будут выполнены действия, содержащиеся в активных шагах. Далее проверяются условия перехода, и, возможно, уже другие шаги становятся активными, но выполняться они будут уже в следующем цикле.

FBD (Function Block Diagram) – это графический язык программирования высокого уровня, обеспечивающий управление потока данных всех типов. Позволяет использовать мощные алгоритмы простым вызовом функций и функциональных блоков. Удовлетворяет непрерывным динамическим процессам. Замечательно подходит для небольших приложений и удобен для реализации сложных вещей подобно ПИД регуляторам, массивам и т. д. Данный язык может использовать большую библиотеку блоков. FBD заимствует символику булевой алгебры и, так как булевы символы имеют входы и выходы, которые могут быть соединены между собой, FBD является более эффективным для представления структурной информации, чем язык релейно-контактных схем.

1.8 Компиляция пользовательской программы

Следующими шагами после создания основных элементов пользовательской программы является его сборка (компиляция и компоновка), передача полученного исполняемого файла на целевое устройство.

Сборка проекта осуществляется с помощью соответствующих кнопок, находящихся на панели инструментов . Для успешного завершения данной операции каждый проект должен иметь как минимум один ресурс (как уже упоминалось, при создании проекта по умолчанию ресурс будет создан). В ресурсе должна быть определена, как минимум, одна задача циклического типа и, как минимум, один экземпляр. Соответственно, проект обязан содержать, как минимум, один программный модуль типа «Программа», причём тело, т.е. алгоритм и логика его выполнения, не может быть пустым (в противном случае будет ошибка компиляции).

Для сборки проекта нажмите кнопку «Сборка проекта в директории сборки».

Рис. 6: *Кнопка сборки проекта*

Результаты сборки выводятся в консоль, расположенную в нижней части окна программы, ошибки сборки выделяются красным цветом. На примере нашего проекта после сборки в консоль выведено сообщение о том, что сборка проведена успешно (подчеркнуто красным цветом).

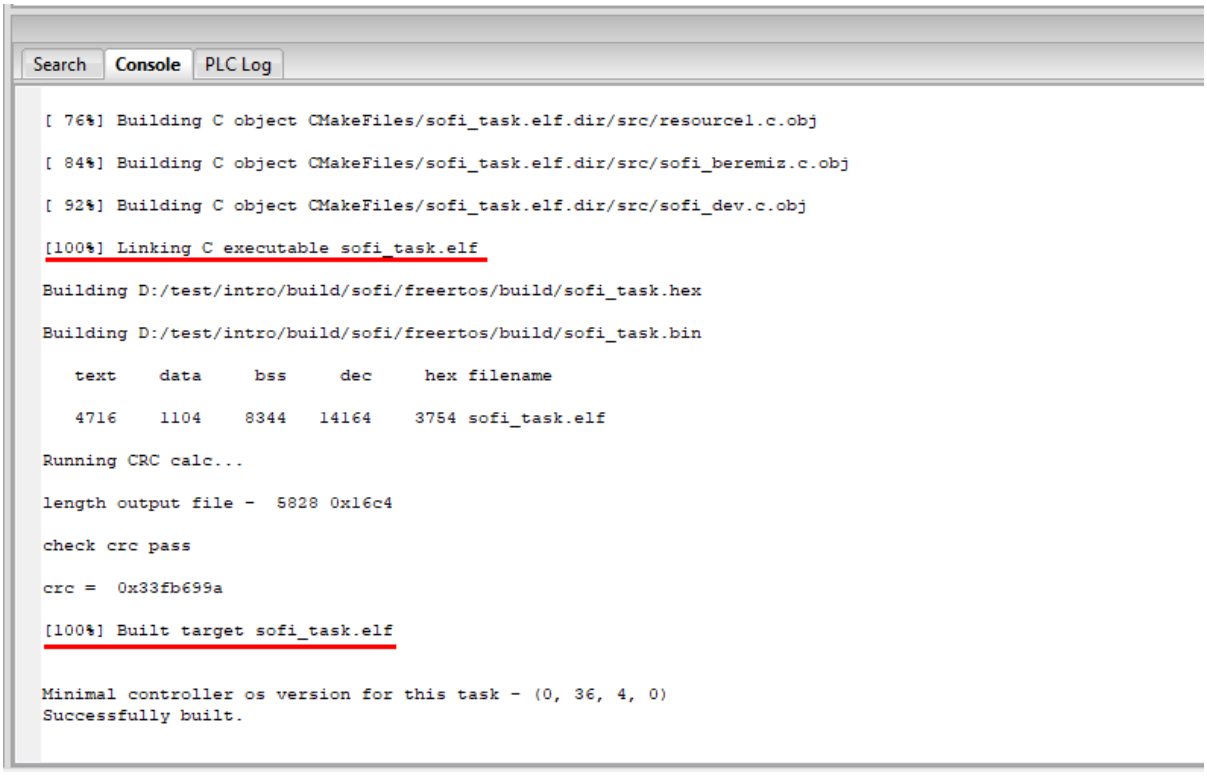


Рис. 7: *Результаты сборки выведены в консоль*

Пересборку проекта можно осуществить, очистив директорию сборки проекта нажатием на кнопку «Очистить директорию сборки проекта». Будет удален сгенерированный на языке ST код проекта и скомпилированный бинарный файл прошивки ПЛК. После этого нажмите кнопку «Сборка проекта в директории сборки», и проект будет собран заново.

Итоговый бинарный файл, который будет загружен в ПЛК, находится в папке проекта с названием *sofi_task_crc.bin*. На рисунке ниже показан путь в папке проекта файла и название файла.

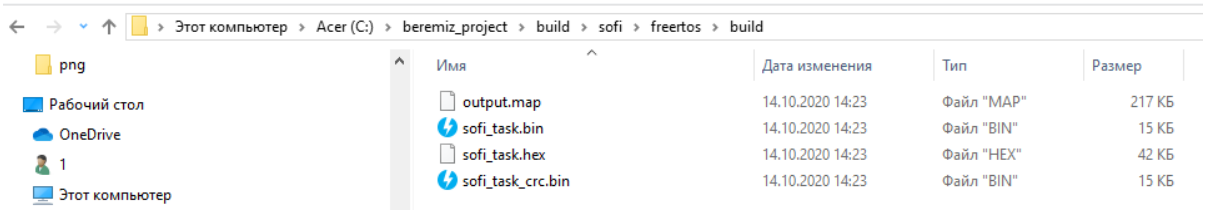


Рис. 8: *Расположение итогового файла проекта*

1.9 Загрузка пользовательской программы в ПЛК BRIC

Загрузка через Beremiz Для загрузки созданноq пользовательской программы необходимо нажать на кнопку «Download task»

Рис. 9: *Кнопка Download task*

Результат загрузки выводится в консоль.

```
downloading task..
used ip address by default http://192.168.1.232
For changing ip address set parameter Project->config->URI_location  - http://192.168.1.232
path D:\WORK\Beremiz_projects\1_lessons\channels\do_state_ctrl/build/sofi/freertos/build/sofi_task_crc.bin
h
Task was stopped
os version in plc - [1, 0, 4, 1]
uploading task
Starting task
reading task state
{'task_state': 9}
User task started
```

Рис. 10: *Результат загрузки проекта в ПЛК*

Описание строк консоли:

downloading task.. – Загрузка пользовательской программы в контроллер;

used ip address by default http://192.168.1.232 – IP-адрес контроллера, куда загружается пользовательская программа;

For changing ip address set parameter Project->config->URI_location - 192.168.1.232 – Заметка для смены IP-адреса контроллера: необходимо зайти в Project->config->URI_location;

path D:WORKBeremiz_projects1_lessonschannelsdo_state_ctrl/build/sofi/freertos/build/sofi_task_crc.b – Размещение итогового скомпилированного файла sofi_task_crc.bin;

Task was stopped – Пользовательская программа в контроллере остановлена;

os version in plc – Версия OS в ПЛК BRIC;

uploading task – Процесс обновления пользовательской программы в контроллере;

Starting task – Процесс запуска пользовательской программы в контроллере;

reading task state {„task_state“: 9} – Чтение статуса пользовательской программы в контроллере;

User task started – Пользовательская программа в контроллере в работе.

Примечание: Консоль показывает процесс компиляции и загрузки пользовательской программы в ПЛК. Наиболее часто встречаемые ошибки данных процессов можно посмотреть здесь.

Загрузка через WEB-страницу ПЛК BRIC

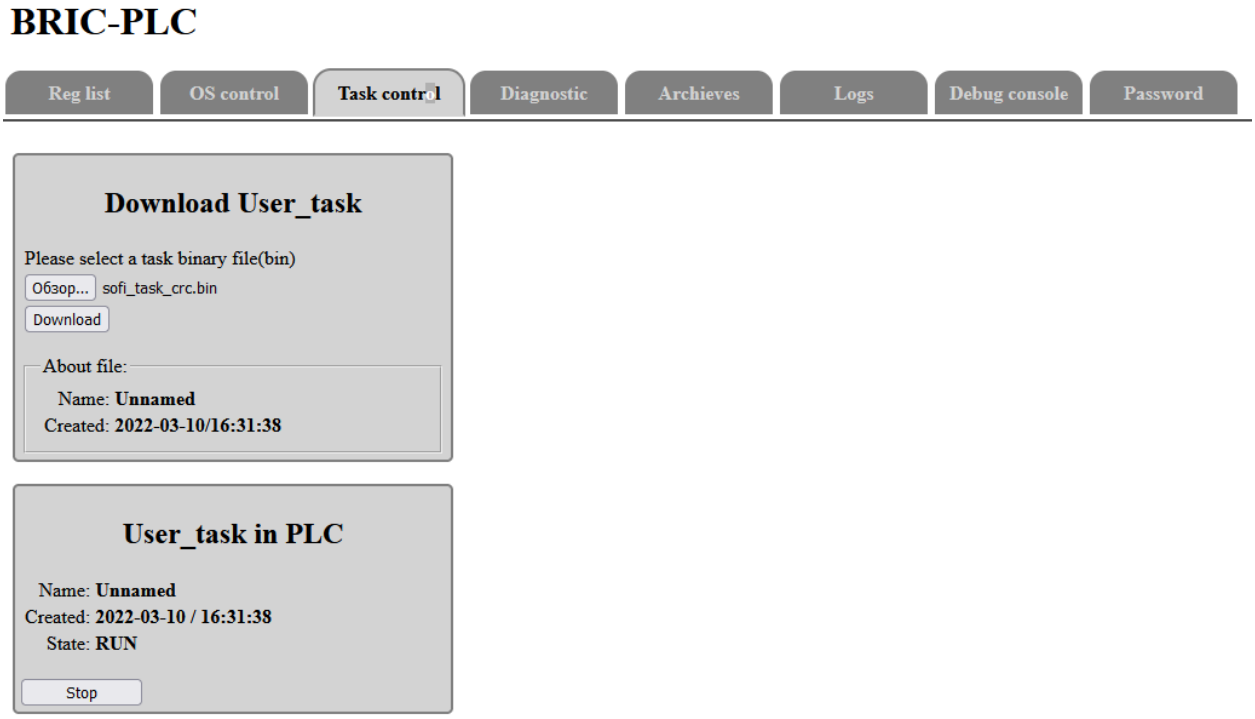
Так же имеется второй способ загрузки *user_task* – через WEB-интерфейс контроллера.

Для загрузки созданного бинарного файла в ПЛК BRIC заходим в браузер по IP-адресу контроллера, по умолчанию: 192.168.1.232

Далее нажимаем на вкладку «Password»(«Пароль») и вводим пароль в поле «Enter password».

Внимание: Паролем по умолчанию является **bric**

После этого переходим во вкладку «Task control»(«Пользоват. программа») и нажимаем на кнопку «Обзор» в поле Download User_task. Выбираем запрашиваемый файл sofi_task_crc.bin. После нажатия кнопки «Download» дожидаемся окончания загрузки. После появления надписи «Download successful, please wait» и возвращения обратно к старнице загрузки нажимаем на кнопку «Run» в поле User_task in PLC. После нажатия пользовательская программа запускается в контроллере, статус можно отследить в строке «State» поля User_task in PLC.



Примечание: При отсутствии возможности подключения к ПЛК BRIC через Ethernet имеется другой способ подключения - USB порт. Адресом порта по умолчанию является 172.16.2.232

Глава 2

Каналы ввода-вывода

2.1 DI. Чтение логического состояния канала

Для прочтения состояния дискретных входов ПЛК BRIC в среде Beremiz разработан функциональный блок READ_DI. Создаём программу на языке FBD. Функциональный блок READ_DI и WRITE_DI добавляются из «Library» во вкладке «DI Function Blocks».

С помощью кнопки VAR из панели навигации добавляем константу «12» для того, чтобы «включить» DO_2 и DO_3.

Примечание:

Про дискретные выходы подробнее объясним в следующих уроках

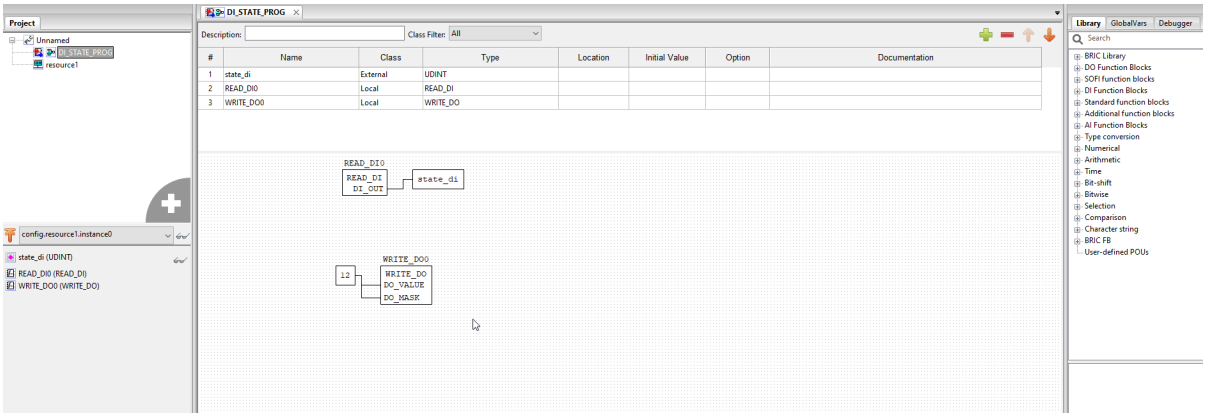


Рис. 1: *Программа для чтения логического состояния*

Класс переменной *state_di* выбирается как «External» с типом данных UDINT. Во вкладку

«GlobalVars» перетаскивается переменная state_di, чтобы она определилась как глобальная в «Config variables» (необходимо левой клавишей мыши зажать столбец «#» для переменной в панели переменных и констант, далее перенести указатель на область GlobalVars и отпустить кнопку мыши (Drag&Drop)).

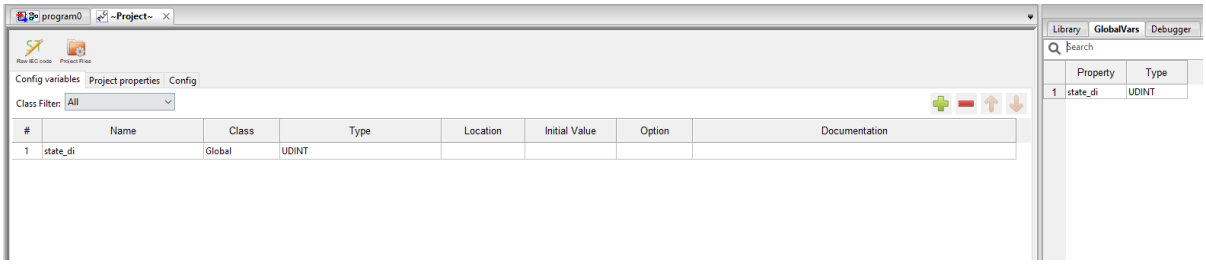


Рис. 2: *Config variables*

Подключаем дискретные каналы DO_2 и DO_3 с каналами DI_9 и DI_10 согласно схеме, представленной ниже.

После загрузки программы в ПЛК BRIC заходим в WEB-страницу контроллера по URL 192.168.1.232 (URL можно поменять в настройках ip address в WEB-странице).

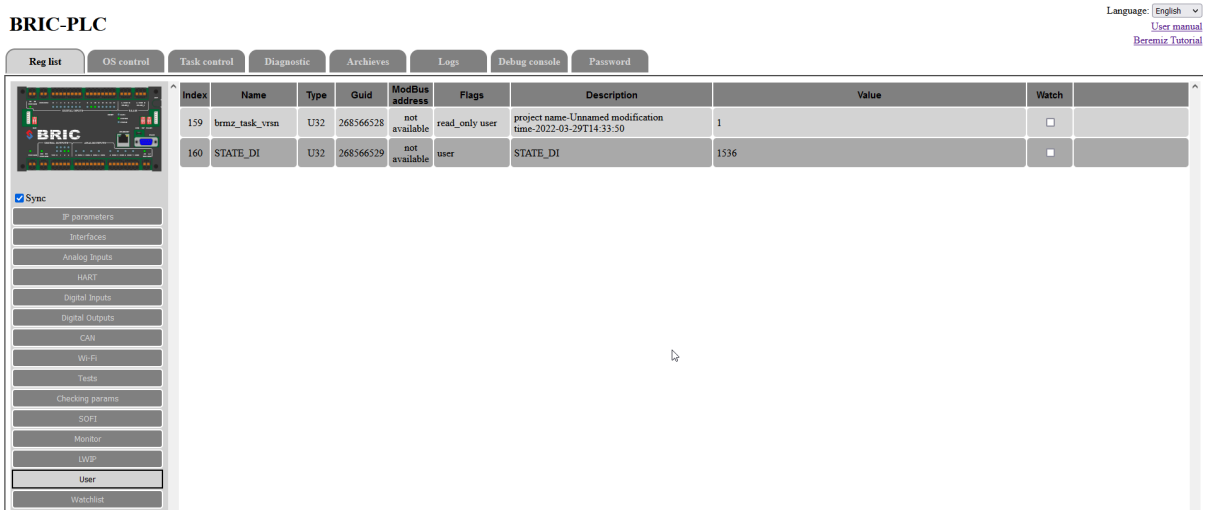


Рис. 3: *WEB-страница ПЛК BRIC*

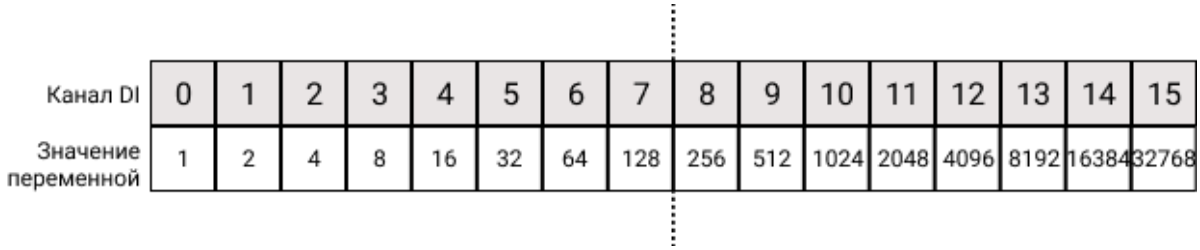
По значению переменой state_di можно определить логическое состояние канала.

Переменная state_di равная числу 1536, которая является суммой значений переменных каналов №9 и №10, на которые пришли дискретные сигналы.

Подсказка: Красные мигающие светодиоды будут означать о том, что в цепи протекает ток ниже 3 mA, подробнее об этом пройдем в дальнейших уроках.

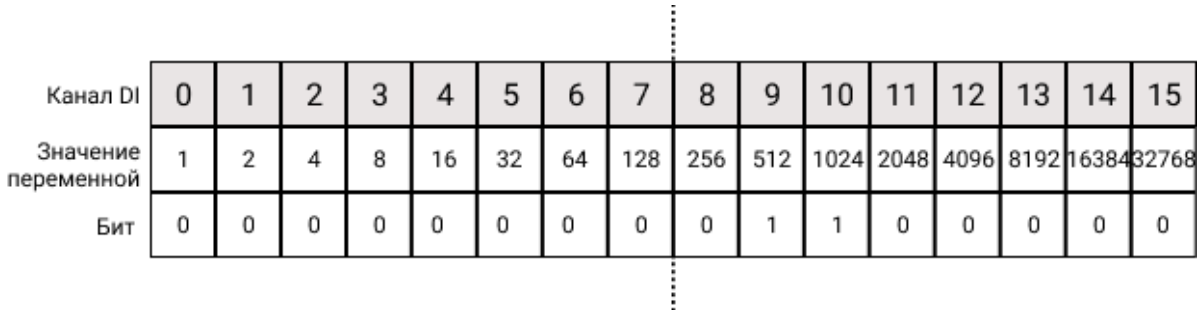
См.также:

Подробно о дискретных входах ПЛК BRIC можно узнать по ссылке



Канал DI	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Значение переменной	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768

Рис. 4: *Определение логического состояния канала*



Канал DI	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Значение переменной	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
Бит	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0

Рис. 5: *Определение логического состояния канала*

2.2 DI. Подсчет импульсов, измерение частоты, тонкая настройка

Для подсчета импульсов дискретных входов ПЛК BRIC в среде Beremiz разработан функциональный блок READ_DI_CNT, а для измерения частоты – READ_DI_FREQ. Создаём программу на языке FBD. Функциональные блоки READ_DI_CNT и READ_DI_FREQ добавляются из «Library» во вкладке «DI Function Blocks».

С помощью кнопки VAR из панели навигации добавляем, например, константу «11», чтобы прочесть состояние с канала DI_11.

Подключаем дискретный канал DO_3 с каналом DI_11 согласно схеме, представленной ниже.

Скомпилируем программу и загрузим в ПЛК. После загрузки программы в ПЛК BRIC заходим в WEB-страницу контроллера по URL 192.168.1.232 (URL можно поменять в настройках ip address в WEB-странице).

Получаем данные количества импульсов и частоты канала DI_11. Изначально они равны «0».

Далее включаем режим ШИМ на канале DO_3 через WEB-страницу контроллера.

Подсказка: Включение режима ШИМ на DO_3

Заходим в раздел Пользовательских переменных и смотрим изменения *di_impulse_cnt* и *di_frequence*

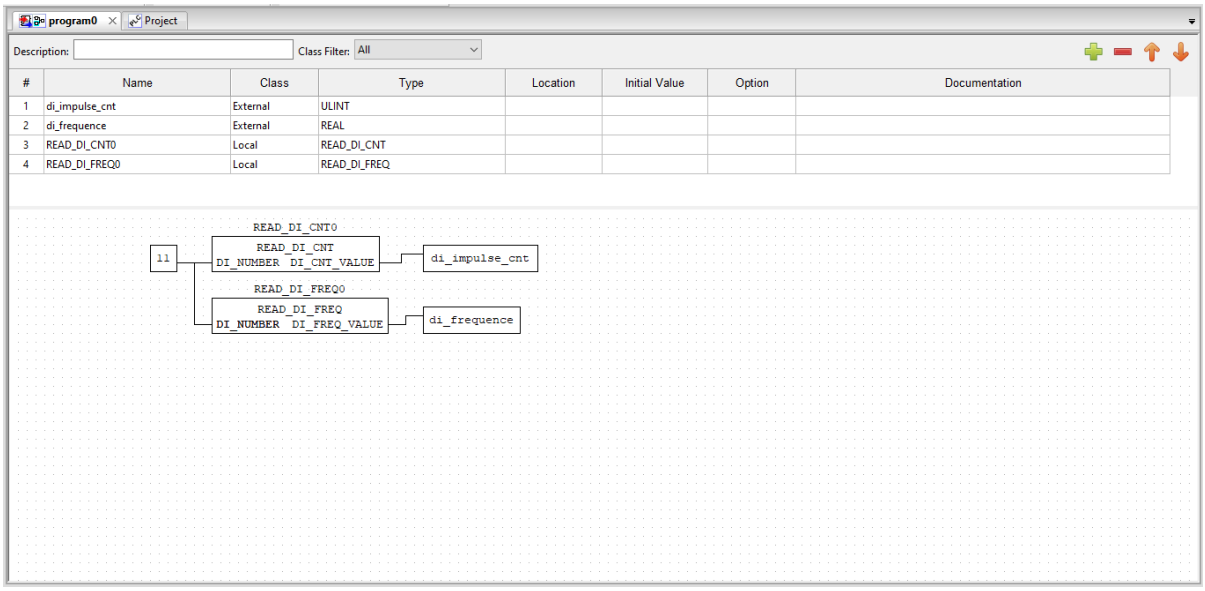


Рис. 6: *Программа для подсчета импульсов и измерения частоты*

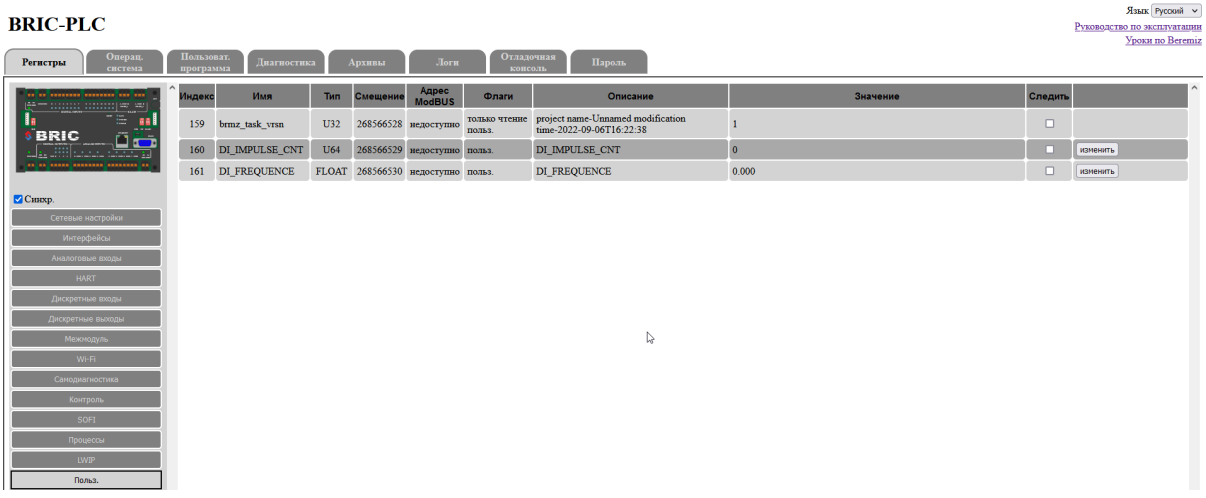


Рис. 7: *Исходные данные количества импульсов и частоты*

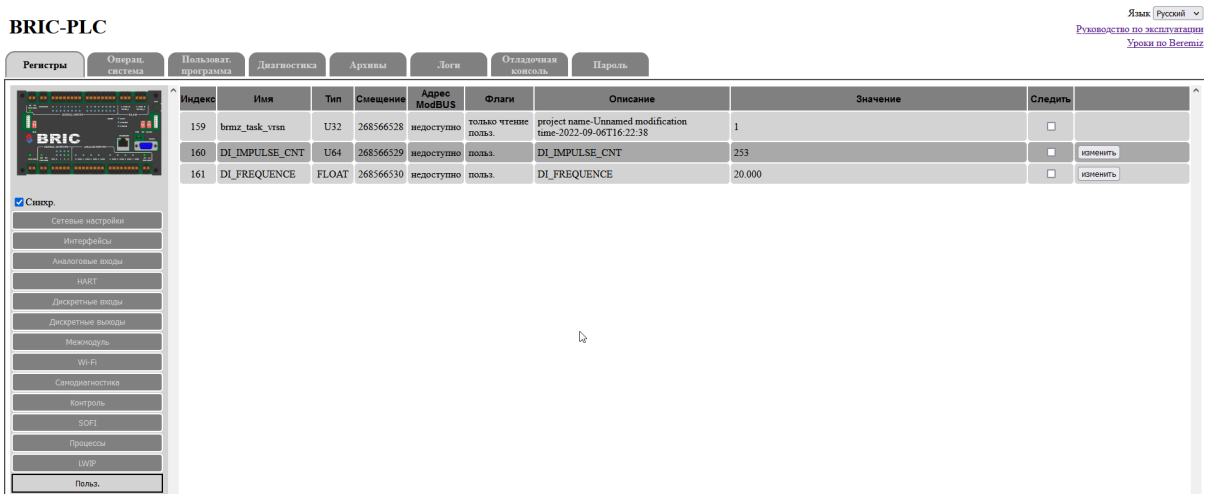


Рис. 8: *Данные количества и частоты после появления импульсного сигнала*

В данном случае количество импульсов инкрментируется с фиксированной частотой 20 Гц.

Рис. 9: *Импульсный сигнал DI_11 на ПЛК BRIC*

Тонкая настройка

В среде Beremiz для ПЛК BRIC разработаны фунциональные блоки WRITE_DI_NOISE_FLTR_10US, WRITE_DI_PULSELESS, WRITE_DI_MODE для тонких настроек дискретных входов.

Функциональный блок WRITE_DI_NOISE_FLTR_10US для указанного дискретного входа задает минимальную длительность входящего импульса. Все что меньше данного значения будет воспринято контроллером как помеха и не будет обрабатываться.

Функциональный блок WRITE_DI_PULSELESS для указанного дискретного входа задает время обнуления измеренной частоты. Если в течение это времени не поступило ни одного импульса, измеренное значение di_freq обнуляется.

Функциональный блок WRITE_DI_MODE для указанного дискретного входа обозначает подключенные опции (0 – не подключены, 1 – подключен счетчик импульсов, 2 – подключен расчет частоты дискретного входа, 3 – подключен счетчик импульсов и расчет частоты дискретного входа).

Создаём программу на языке FBD. Для канала DI_0 установим минимальную длительность импульса 3*10мкс = 30 мкс. Для DI_1 установим время обнуления измерения частоты - 20000 мс (20 сек). Для DI_2 запишем код подключенных функций данного канала - 1 (счетчик импульсов).

Результат загруженной программы можно посмотреть в WEB-странице ПЛК во вкладке «Дискретные входы (Discrete inputs)». Как видно на рисунках ниже, изменения внесены:

См.также:

Подробно о дискретных входах ПЛК BRIC можно узнать по ссылке

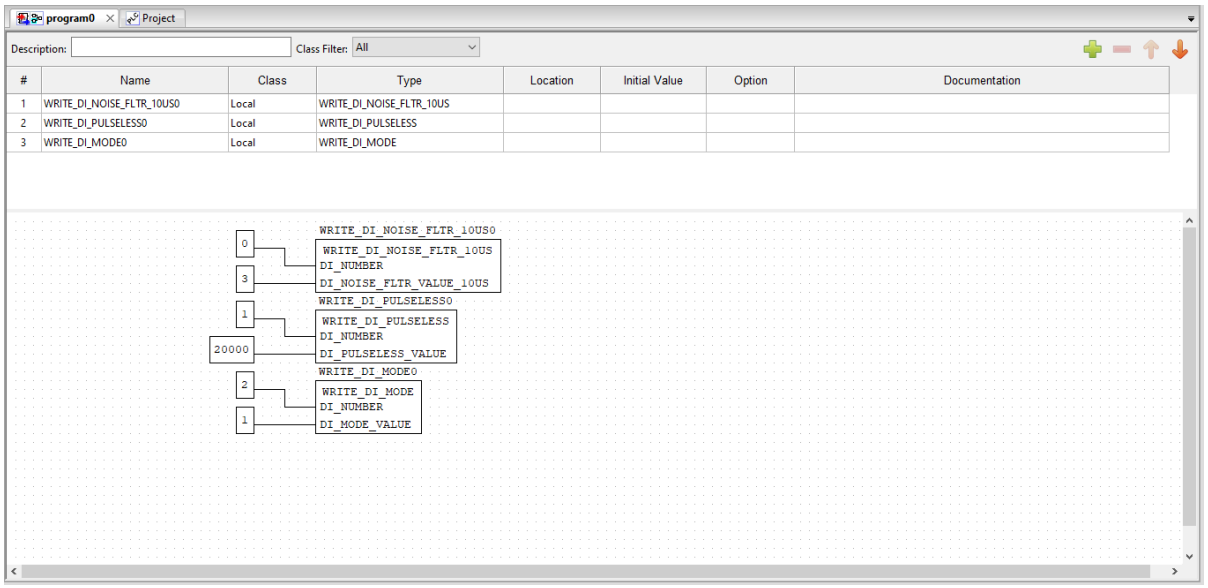


Рис. 10: *Программа для тонкой настройки дискретных входов*

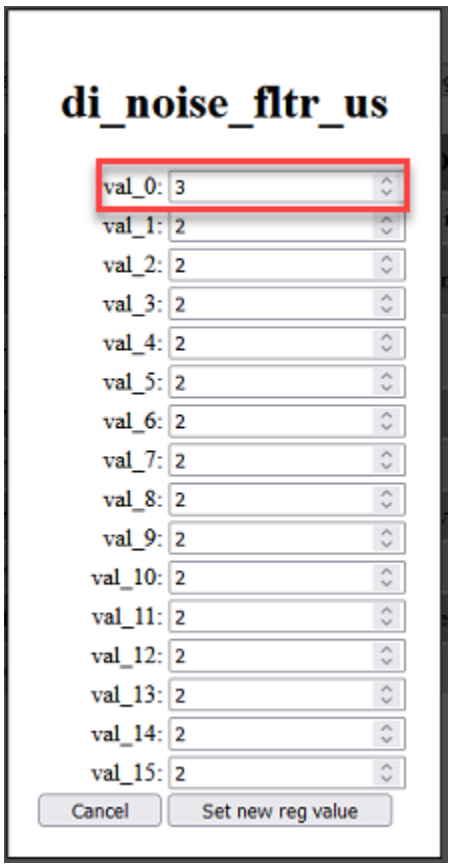


Рис. 11: *Период нечувствительности импульса*

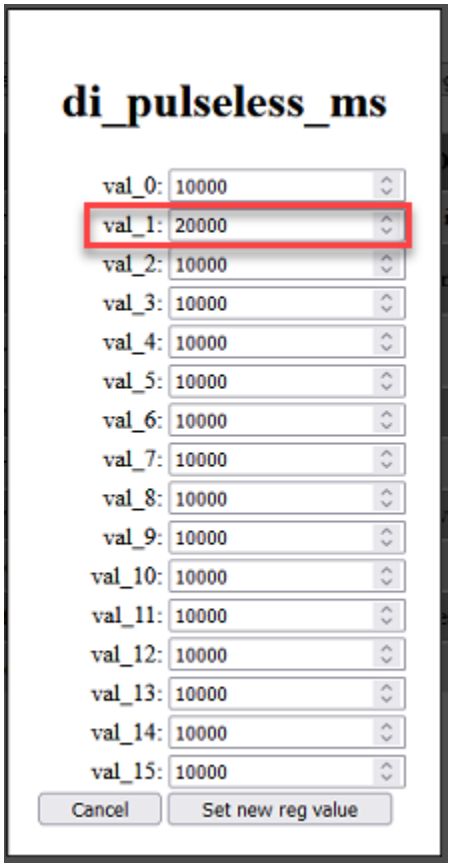


Рис. 12: *Время обнуления измеренной частоты*

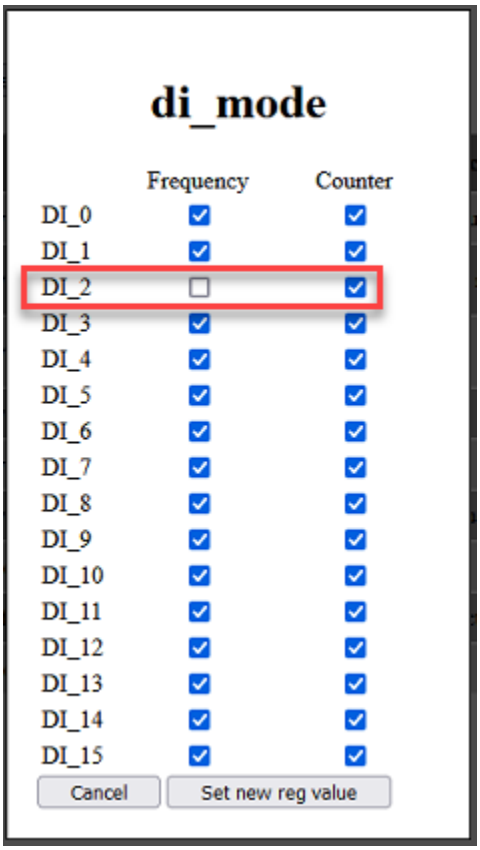
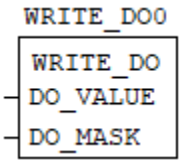


Рис. 13: *Подключенная функция - подсчет импульсов*

2.3 DO. Управление логическим состоянием. Независимое управление несколькими каналами

Для записи значения состояния дискретных выходов ПЛК BRIC в Beremiz реализован функциональный блок WRITE_DO.



Данный функциональный блок имеет 2 входа: DO_VALUE (значение состояния дискретных выходов) и DO_MASK (маска, разрешающая изменять состояние дискретных выходов). Выходы прописываются побитово:

Канал DO	0	1	2	3
Значение переменной	1	2	4	8

Рис. 14: *Определение логического состояния канала*

То есть, чтобы «включить» дискретные выходы, например, DO_2 и DO_3 необходимо прописать в DO_VALUE и DO_MASK значение 12. Создаём программу на языке FBD. Функциональный блок WRITE_DO добавляется из «Library» во вкладке «DO Function Blocks».

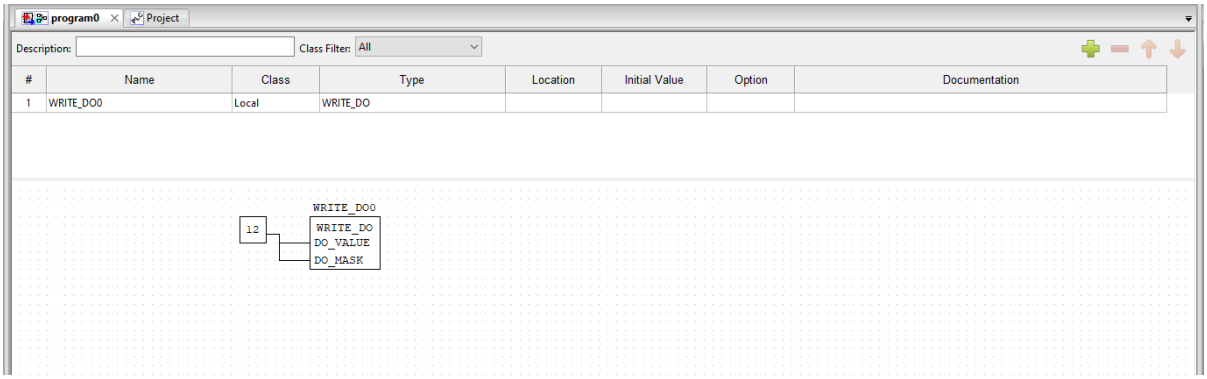


Рис. 15: *Программа на FBD для записи состояния DO_2 и DO_3*

Подключаем резисторы для исключения сигнализации обрыва на каналах. После загрузки программы в ПЛК загораются зеленые светодиоды на дискретных выходах DO_2 и DO_3.

Важно: DO_VALUE имеет тип данных UINT с диапазоном от 0. . . 65535. В случае с Beremiz участвует младший полубайт, а старший отвечает за маскирование. Рекомендуется использовать диапазон 0..15.

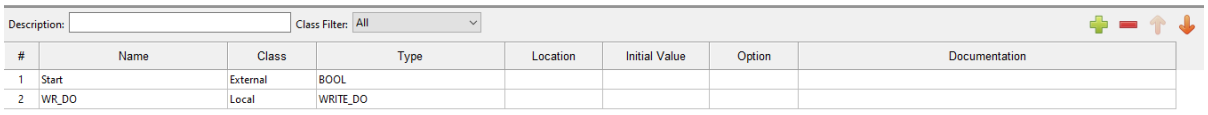
Для того, чтобы подробнее узнать о независимом управлении несколькими дискретными выходами напишем проект. В него буду входить две программы на языке ST. Допустим, что при появлении

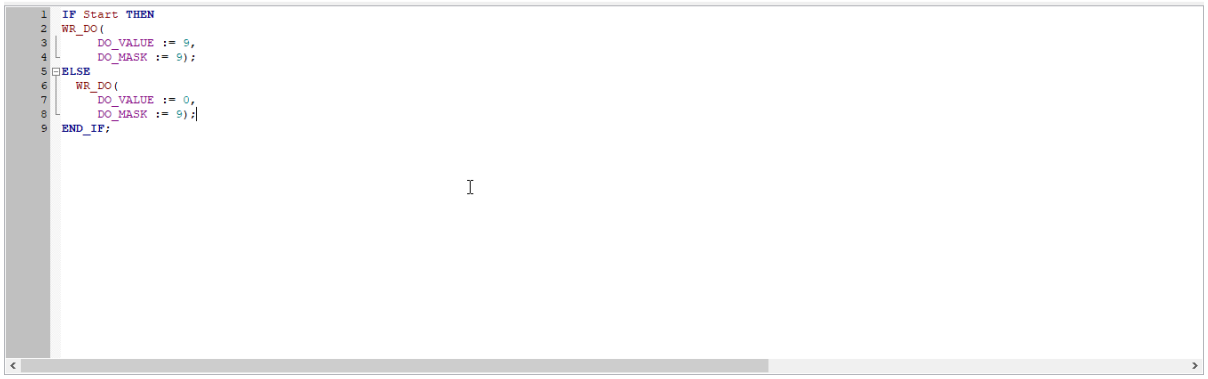
Канал DO	0	1	2	3
Значение переменной	1	2	4	8
DO_VALUE	0	0	1	1
DO_MASK	0	0	1	1

Рис. 16: *Запись состояния дискретного выхода*

Рис. 17: *DO_2 и DO_3 на ПЛК BRIC*

одного сигнала, например, «start» должны включиться выходы DO_0 и DO_3. А при появлении сигнала «start2» должен включиться дискретный выход DO_2. При отсутствии данных сигналов каналы должны выключиться. Допустим первая наша программа будет отвечать за управление дискретных выходов DO_0 и DO_3. Программа будет иметь вид, представленный на рисунке ниже.





Вторая программа, отвечающая за дискретный выход DO_2 представлена на рисунке ниже. Дискретный выход DO_2 управляется внешним сигналом «start2».

Добавляем программы в ресурсы, компилируем. Далее загружаем наш проект в контроллер. Заходим в WEB-страницу и открываем вкладку «USER». При изменении переменной «start» из 0 в 1 - дискретные выходы DO_0 и DO_3 «включаются».

При изменении переменной «start2» из 0 в 1 включается дискретный выход DO_2. Заметьте, что включенные дискретные каналы DO_0 и DO_3 остаются без изменений.

Поставленная задача решена, для того, чтобы выключить все дискретные выходы контроллера в данном случае необходимо «выключить» сигналы «start» и «start2».

Для чтения значения состояния дискретных выходов ПЛК BRIC в Beremiz реализован функциональный блок READ_DO (при включенном канале DO логическая 1 информирует о протекании в цепи

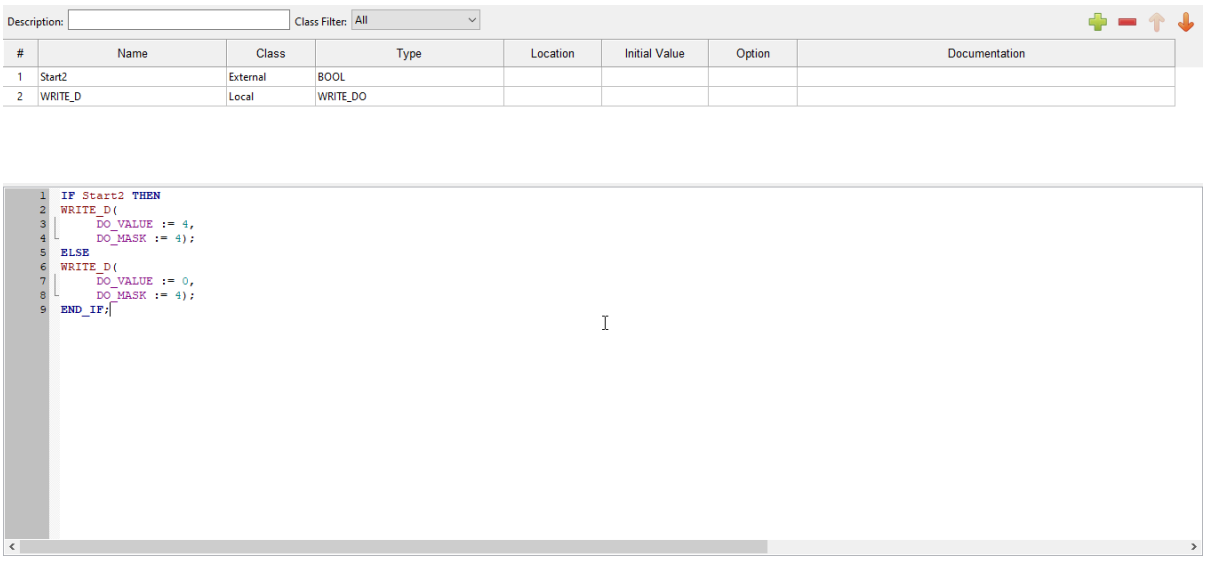


Рис. 18: *DO_0 и DO_3 на ПЛК BRIC*

тока более 2 мА, а логический 0 – об обрыве цепи). Добавим функциональный блок READ_DO и переменную do_read в предыдущую программу.

Если в каком-либо из задействованных каналов будет обрыв или короткое замыкание, то значение переменной do_read поменяется.

См.также:

Подробно о дискретных выходах ПЛК BRIC можно узнать по ссылке

2.4 DO. Управление каналом DO по состоянию канала DI

Для данного урока подготовим ПЛК BRIC: соединяем DO_GND и DI_COM, DO_0 -> DI_0, DO_1 -> DI_1.

Подсказка: В данном случае каналы DO работают как сухой контакт

Напишем программу на языке ST для управления каналами DO_0 и DO_1 по состоянию дискретных входов. В переменную *di_out* записывается состояние дискретных входов (READ_DI). Реализация программы представлена ниже:

Данная программа записывает вначале цикла значение логической «1» в канал DO_0 с разрешающей маской «15»(все каналы). Так как DO_0 соединен с DI_0 состояние DI_STATE будет равно «1». Далее программа проверяет состояние дискретных входов, если оно равно «1», то записываем в канал DO_1 логическую «1» (DO_VALUE = 2). Таким же образом проверяется состояние каналов DI, и если оно равно «2», то цикл начинается заново. Цикл программы сделаем равным 1 секунде.

Рис. 19: *Состояние дискретных выходов ПЛК BRIC*

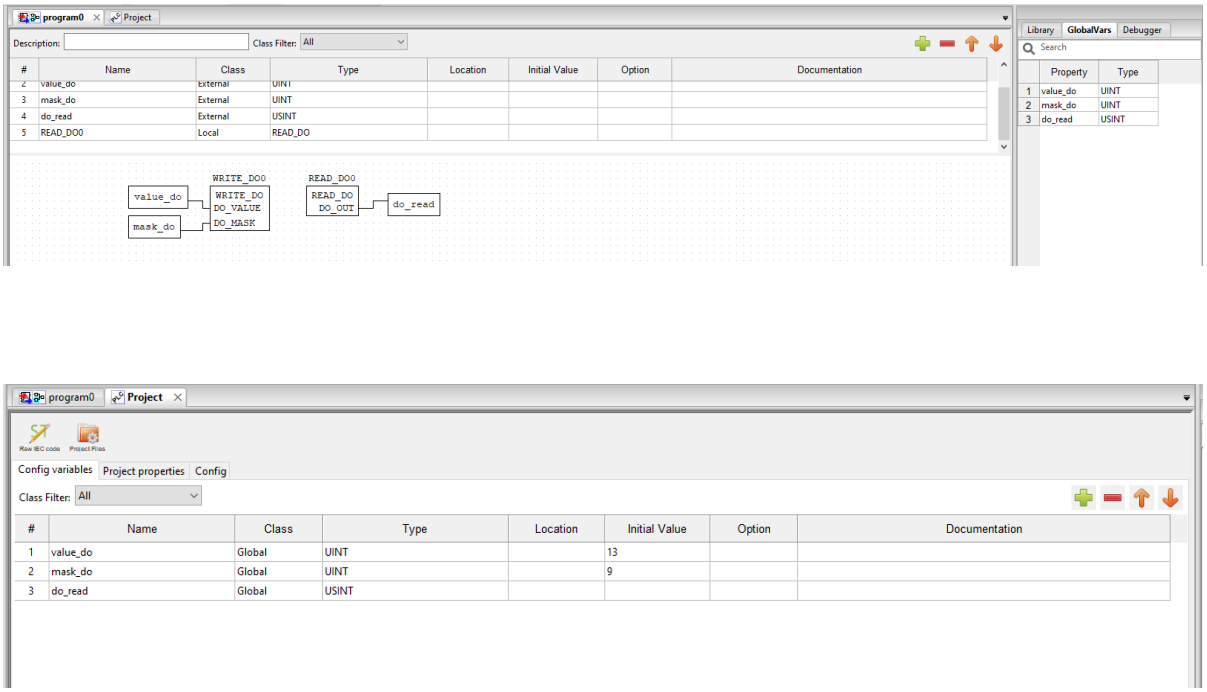


Рис. 20: *Программа для чтения состояния дискретных каналов*

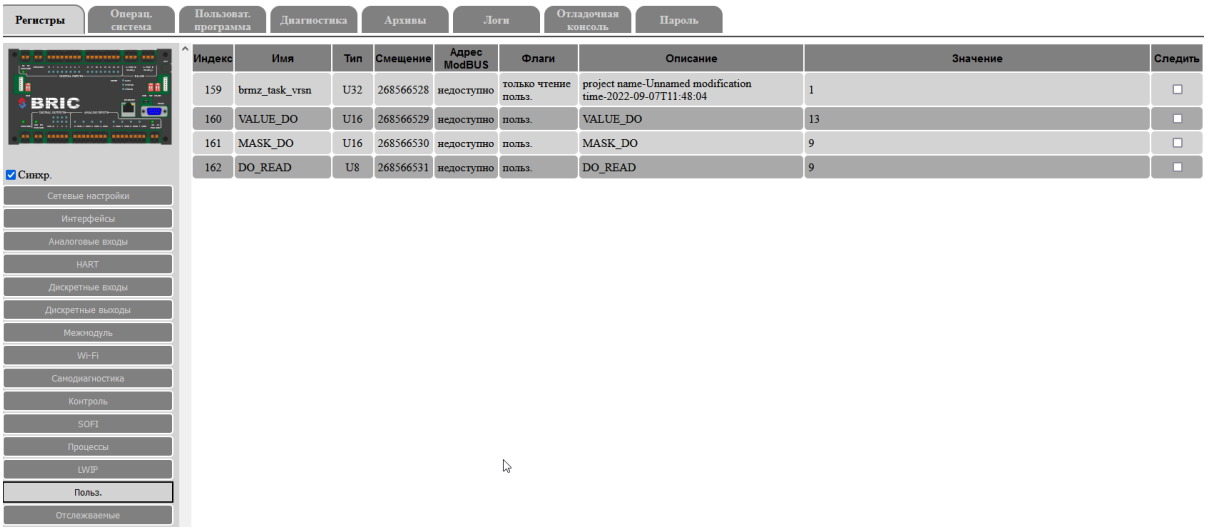


Рис. 21: *Результат вывода программы в WEB-странице*

Рис. 22: *Подключение каналов ПЛК BRIC*

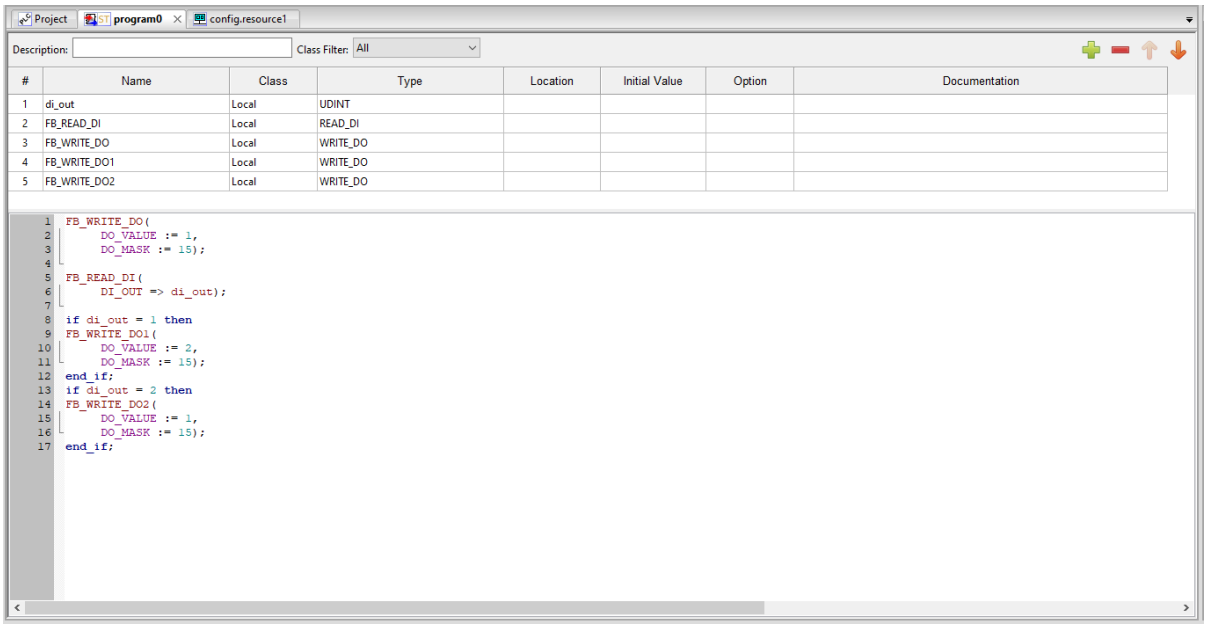


Рис. 23: *Программа на языке ST*

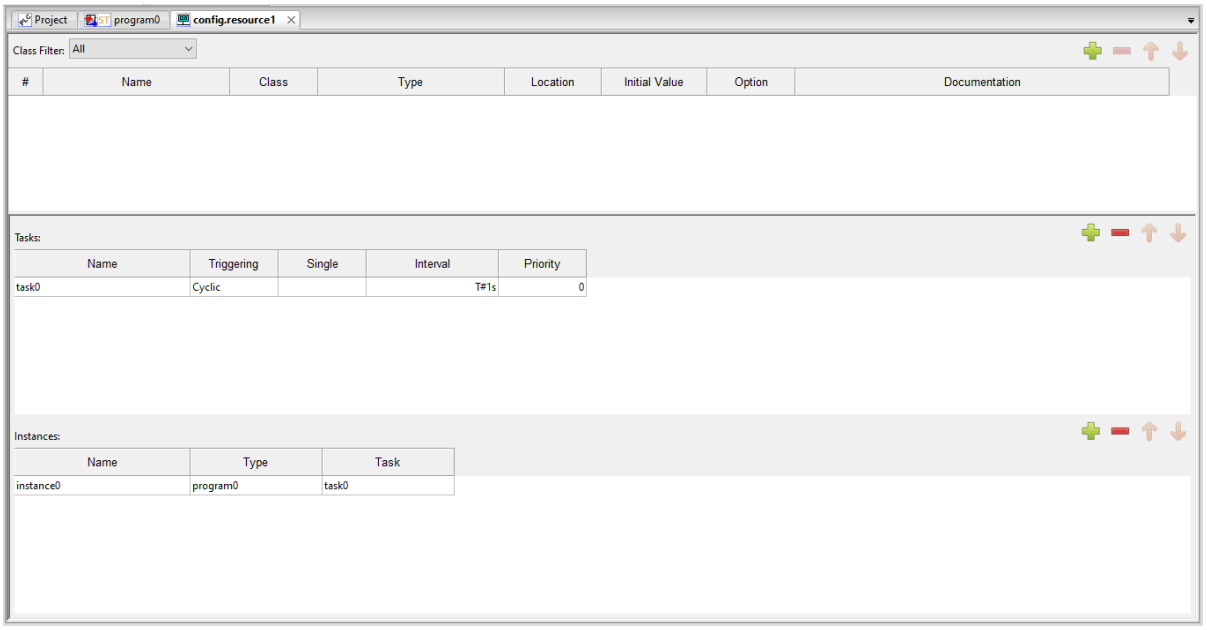


Рис. 24: *Цикл программы*

После загрузки прошивки в контроллер с цикличностью в 1 секунду поочередно начинают гореть зеленые индикаторные светодиоды каналов DO_0-DI_0 и DO_1-DI_1.

Рис. 25: *ПЛК BRIC после реализации программы*

Таким образом, в данной программе видно, что состояние каналов DO меняется при изменении состояния каналов DI.

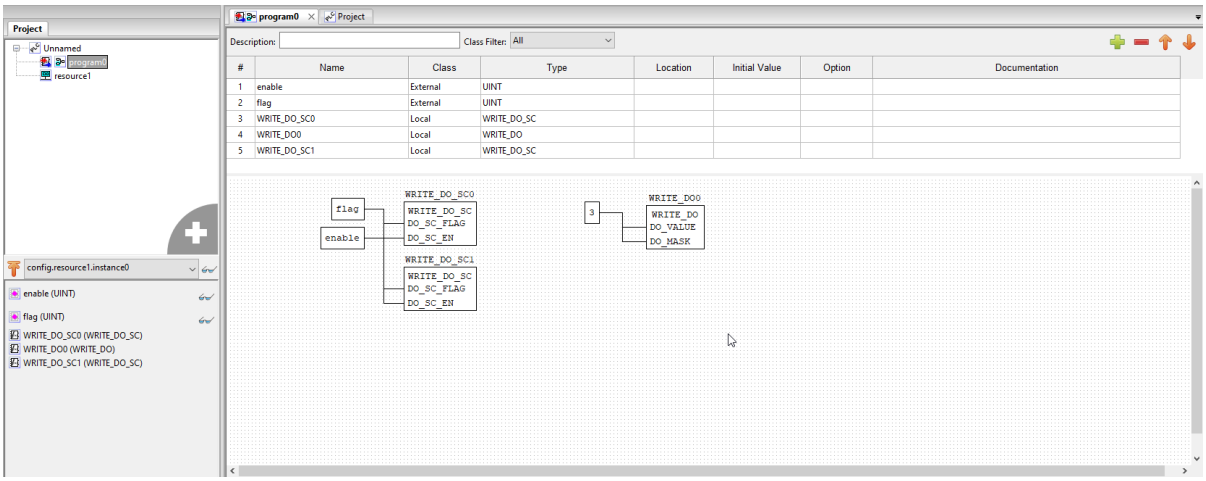
См.также:

Подробно о дискретных выходах ПЛК BRIC можно узнать по ссылке

2.5 DO. Обратная связь (обрыв, короткое замыкание), управление защитой от короткого замыкания

В Beremize для управления защитой от короткого замыкания дискретных выходов ПЛК BRIC реализован функциональный блок WRITE_DO_SC. При наличии короткого замыкания загорается красный индикаторный светодиод, соответствующий канал DO отключается, управление каналом блокируется и выставляется флаг DO_SC. При наличии обрыва на линии дискретных выходов красный светодиод мигает.

Напишем программу на языке FBD. В ней мы будем проверять дискретные выходы DO_0 и DO_1 на КЗ. Программа выглядит следующим образом.



Если на линии выходов DO_0 и DO_1 нет обрывов, то загораются зеленые индикаторы на каналах.

Рис. 26: *Визуализация программы на ПЛК*

Включаем программную защиту от короткого замыкания с помощью переменной «enable» .

Далее делаем КЗ одновременно на двух каналах DO_0 и DO_1 с помощью перемычки, замыкая их на «DO_VCC». В результате срабатывает программная защита, каналы отключаются и устанавливается флаги SC Flag, загораются красные светодиоды и управление соответствующими каналами блокируется.

Сбросить флаг можно записав в переменную «Flag» значение «0».

Index	Name	Type	Guid	ModBus address	Flags	Description	Value	Watch	
150	brmz_task_vrsn	U32	268566528	not available	read_only user	project name-Unnamed modification time-2022-03-04T16:51:47	1	☐	
151	ENABLE	U16	268566529	not available	user	ENABLE	0	☐	change
152	FLAG	U16	268566530	not available	user	FLAG	0	☐	change

Рис. 27: *Web-страница программы*

Рис. 28: *Визуализация программы на ПЛК*

Если оборвать цепь на каком либо задействованном дискретном выходе, то красный светодиод соответсвующего канала начинает мигать. В нашей программе протестируем на канале DO_0.

Рис. 29: *Визуализация программы на ПЛК*

См.также:

Подробно о дискретных выходах ПЛК BRIC можно узнать по ссылке

2.6 DO. «Бегущий огонь» на каналах DO

Для реализации «Бегущего огня» в ПЛК BRIC напишем программу на языке ST. Добавим локальную переменную *k* с типом данных *USINT*. Также добавим функциональный блок *WRITE_DO_PWM_CTRL* для режима ШИМ дискретных выходов.

Переменная *k* инкрементируется, при достижении 4 она приравнивается к нулю. В данной программе используется элемент **CASE**, для каждого значения *k(0..4)* прописаны действия:

- при k = 1 запускается режим ШИМ для нулевого канала и выключается для третьего;
- при k = 2 запускается режим ШИМ для первого канала и выключается для нулевого и так далее;

Цикл программы сделаем равным 1 секунде.

Визуализация программы представлена ниже:

См.также:

Подробно о дискретных выходах ПЛК BRIC можно узнать по ссылке

2.7 DO - PWM. Настройки и управление

Для настройки и управления PWM в Beremize реализованы функциональные блоки WRITE_DO_PWM_FREQ и WRITE_DO_PWM_CTRL. С помощью функционального блока WRITE_DO_PWM_CTRL можно выбирать нужный канал, менять скважность и включать ШИМ на выбранном канале, а с помощью WRITE_DO_PWM_FREQ – частоту. Любой канал может работать в режиме широтно-импульсной модуляции (ШИМ) на частоте от 20 Гц до 10 кГц, частота общая для всех каналов DO. В режиме ШИМ параметры PWM Frequency и PWM Duty пересчитываются в количество тактов и загружаются в соответствующие регистры опорного таймера, тактируемого частотой 1 МГц. В начальный момент времени соответствующий канал DO включается, а по достижении таймера значения соответствующего длительности PWM Duty канал DO выключается. При достижении таймером значения периода рабочей частоты таймер обнуляется и процесс повторяется сначала. Переключение канала DO в режиме ШИМ происходит без участия процессора. Возможно

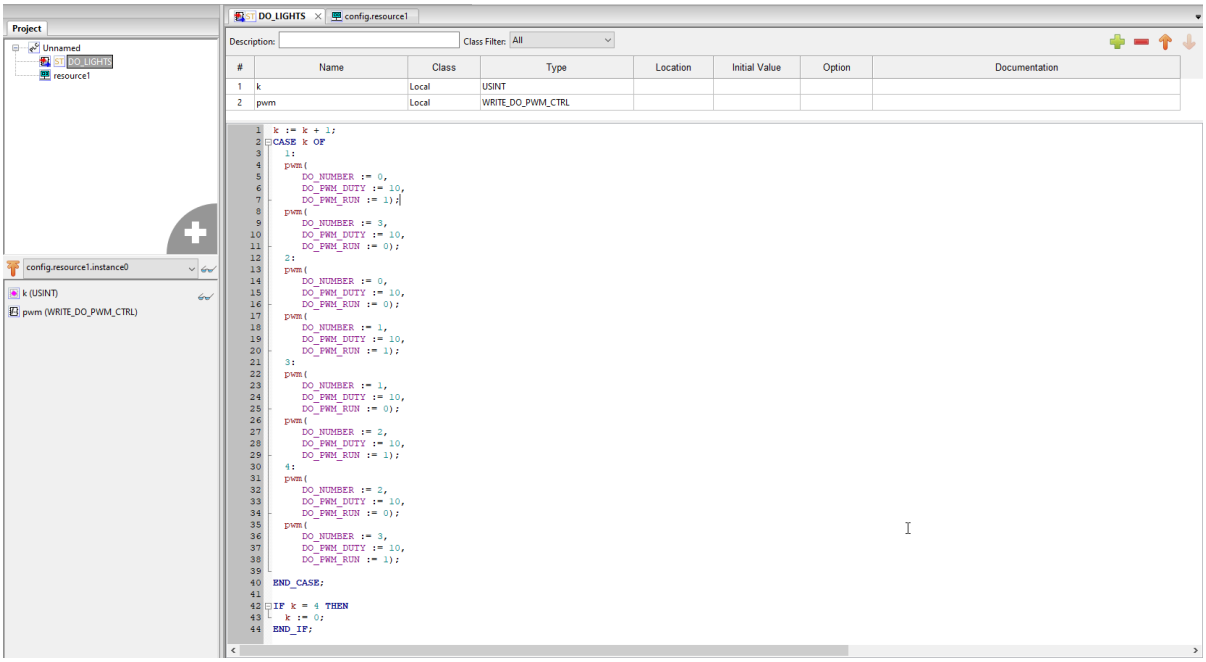


Рис. 30: *Программа «Бегущий огонь» на языке ST*

Рис. 31: *«Бегущий огонь» на каналах DO в ПЛК BRIC*

изменение скважности без остановки ШИМ изменением значения PWM Duty. Напишем программу на языке FBD для изменения скважности канала DO_0 и задания частоты ШИМ, общей для всех каналов.

После загрузки прошивки в ПЛК заходим в WEB-страницу, задаем скважность и частоту.

Результаты можно увидеть во вкладке Digital Outputs: PWM control и PWM frequency Hz.

См.также:

Подробно о дискретных выходах ПЛК BRIC можно узнать по ссылке

2.8 AI. Описание регистров канала

Для прочтения состояния аналоговых входов ПЛК BRIC в среде Beremiz разработаны функциональные блоки READ_AI, READ_AI_STATE и READ_AI_REAL.

Создаём программу на языке FBD. Функциональные блоки добавляются из «Library» во вкладке «AI Function Blocks».

Данная программа прочитывает состояние AI_0, а так же с помощью READ_AI_STATE можем узнать состояние всех аналоговых входов. Каждый канал имеет индикаторный светодиод, отображающий состояние канала. Чем выше частота моргания светодиода – тем больше измеряемая величина. После подачи сигнала на AI_0 появляются значения *ai_val*, *state_val* и *ai_real*.

Наша переменная *ai_val* – это параметр AI_UINT_X, результат измерения аналогового канала в единицах АЦП в диапазоне 0...16383. В данном случае *ai_val* равно 12895. Переменная *ai_real* – это параметр AI_PHYSICAL_X, результат измерения аналогового канала в физических единицах в диапазоне 0...20.0. В нашем случае *ai_real* равно 17.9 mA. Переменная *state_val* – это параметр

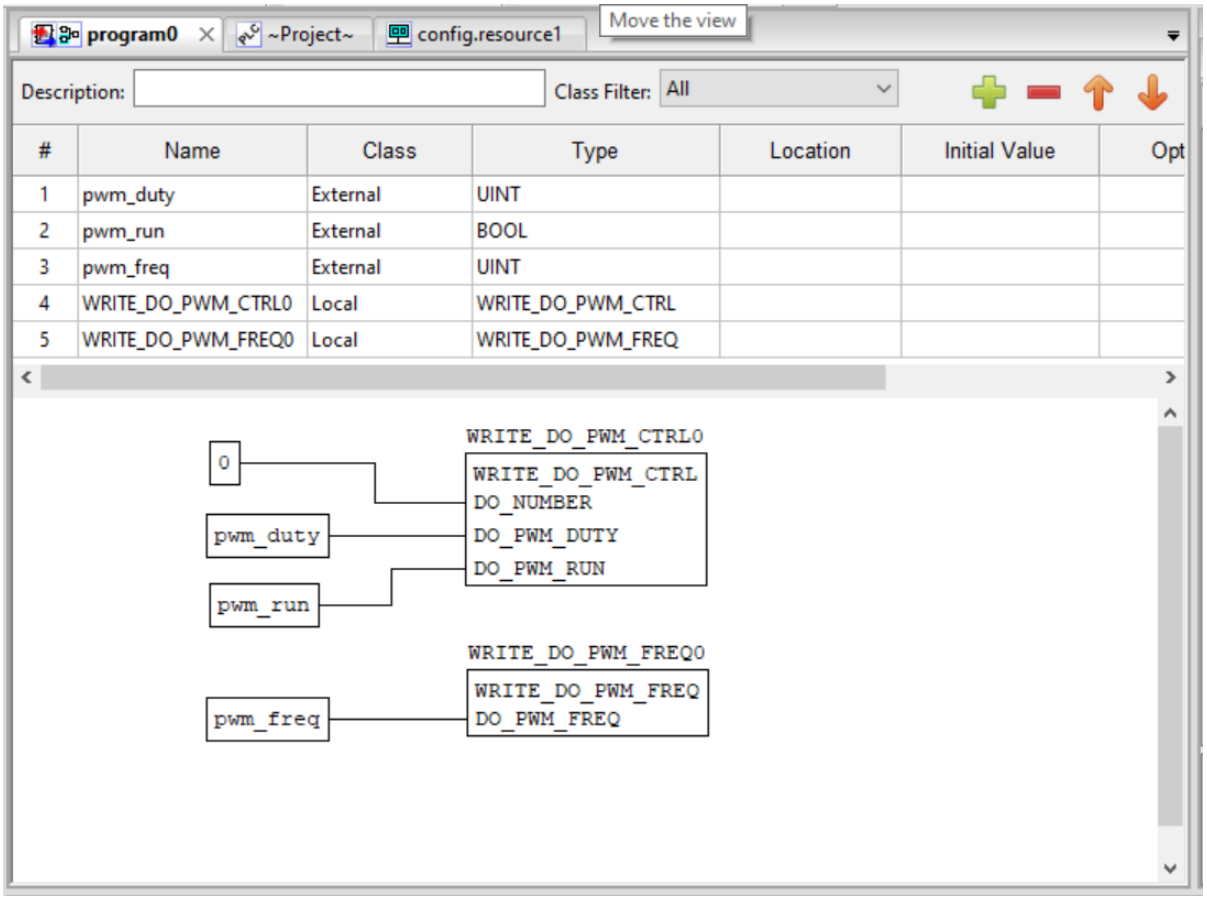


Рис. 32: *Программа на языке FBD*

Рис. 33: *WEB-страница контроллера*

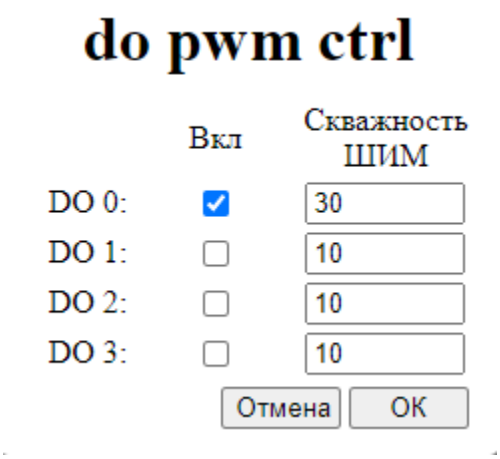


Рис. 34: *PWM control*

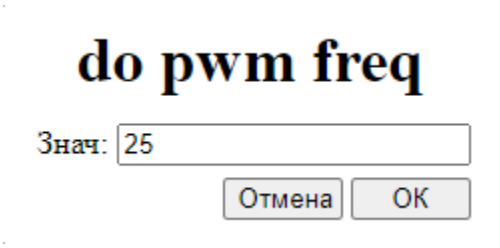


Рис. 35: *PWM frequency*

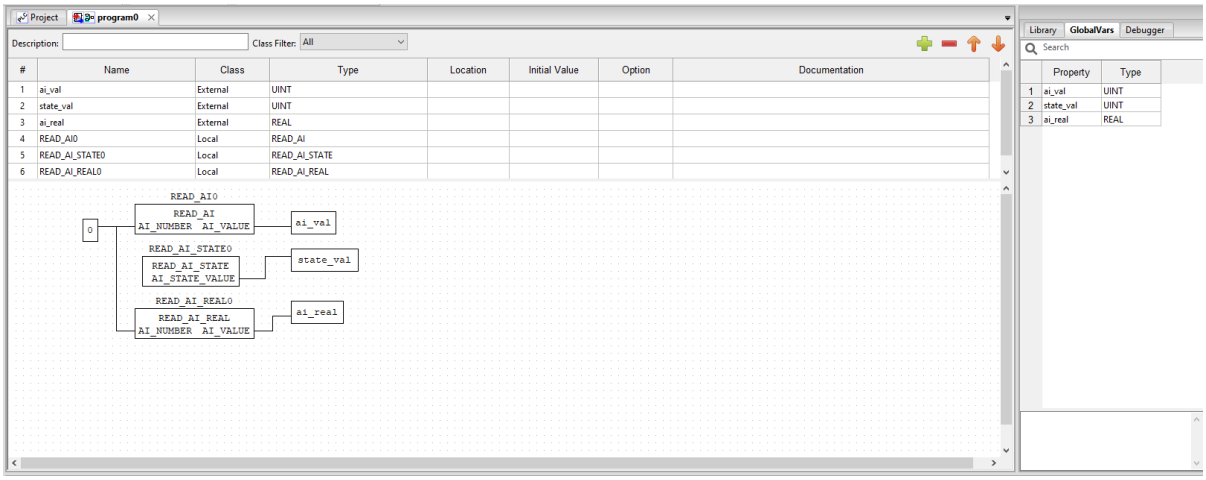
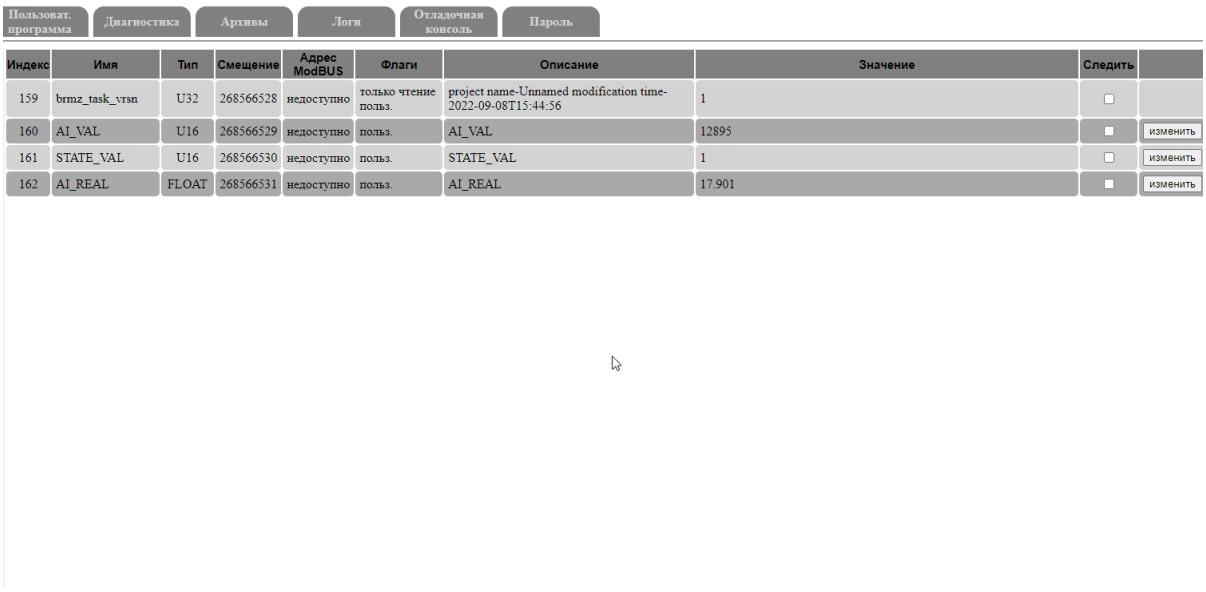


Рис. 36: *Программа для чтения логического состояния аналогового входа AI_0*



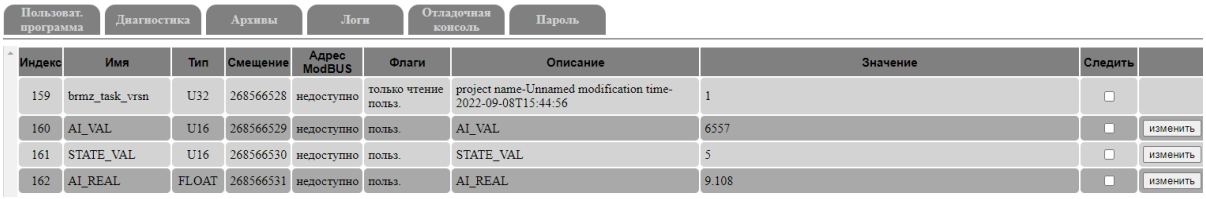
Индекс	Имя	Тип	Смещение	Адрес ModBUS	Флаги	Описание	Значение	Следить	
159	brmz_task_vrsn	U32	268566528	недоступно	только чтение польз.	project name-Unnamed modification time-2022-09-08T15:44:56	1	☐	
160	AI_VAL	U16	268566529	недоступно	польз.	AI_VAL	12895	☐	изменить
161	STATE_VAL	U16	268566530	недоступно	польз.	STATE_VAL	1	☐	изменить
162	AI_REAL	FLOAT	268566531	недоступно	польз.	AI_REAL	17.901	☐	изменить

Рис. 37: *WEB-страница контроллера*

Рис. 38: *Аналоговый сигнал на AI_0 в ПЛК BRIC*

AI_STATE, состояние канала. Логическая единица – измеренное значение тока лежит в диапазоне 4 - 20 мА, логический ноль – измеренное значение ниже 4 mA либо выше 20 mA. В данном случае задействован только AI_0, поэтому *state_val* равно 1.

Чтобы посмотреть на изменения глобальных параметров снизим величину аналогового сигнала на AI_0, а также подключим канал AI_2.



Индекс	Имя	Тип	Смещение	Адрес ModBUS	Флаги	Описание	Значение	Следить	
159	brmz_task_vrsn	U32	268566528	недоступно	только чтение польз.	project name-Unnamed modification time-2022-09-08T15:44:56	1	☐	
160	AI_VAL	U16	268566529	недоступно	польз.	AI_VAL	6557	☐	изменить
161	STATE_VAL	U16	268566530	недоступно	польз.	STATE_VAL	5	☐	изменить
162	AI_REAL	FLOAT	268566531	недоступно	польз.	AI_REAL	9.108	☐	изменить

Рис. 39: *WEB-страница контроллера после внесений изменений*

Переменная *ai_val* снизится до 6557, что равнозначно *ai_real* – 9.1 mA. Значение *state_val* будет равно 5, что означает задействование каналов AI_0 и AI_2.

См.также:

Подробно о аналоговых входах ПЛК BRIC можно узнать по ссылке

Рис. 40: *Аналоговые сигналы на AI_0 и AI_2 в ПЛК BRIC*

Канал AI	0	1	2	3	4	5	6	7
Значение переменной	1	2	4	8	16	32	64	128

Рис. 41: *Значения переменных аналоговых входов*

2.9 AI. Подключение датчика температуры 4-20 мА. Преобразование в инженерные единицы с масштабированием

Для данного урока необходим датчик температуры с унифицированным выходным сигналом 4-20mA. Аналоговые каналы измерения тока могут быть выполнены как в пассивном, так и в активном исполнении. Все виды подключения к ПЛК BRIC показаны тут.

В нашем случае будем использовать датчик температуры ТПУ 0304/М2-Н фирмы ЭЛЕМЕР с заводской установкой НСХ Pt100 dt:(-50...150)℃ с внешним источником питания, канал AI_1 - пассивный.

Рис. 42: *Подключение датчика температуры ТПУ 0304/М2-Н фирмы ЭЛЕМЕР к ПЛК*

Напишем программу на языке FBD. Для масштабирования переменные *scale_end* и *scale_start* – начальное и конечное значение физической шкалы, объявлены как глобальные. Расчет значения физической **линейной шкалы** в зависимости от значения унифицированного сигнала равна:

$$d = ((sgv - sgs)/(sge - sgs)) * (sce - scs) + scs,$$

где **d** - результат;

sgv - значение унифицированного сигнала;

scs - начальное значение физической шкалы;

sce - конечное значение физической шкалы;

sgs - начальное значение унифицированного сигнала;

sge - конечное значение унифицированного сигнала.

Как видно по перменной *temperature* значения с датчиком температуры совпадают.

См.также:

Подробно о аналоговых входах ПЛК BRIC можно узнать по ссылке

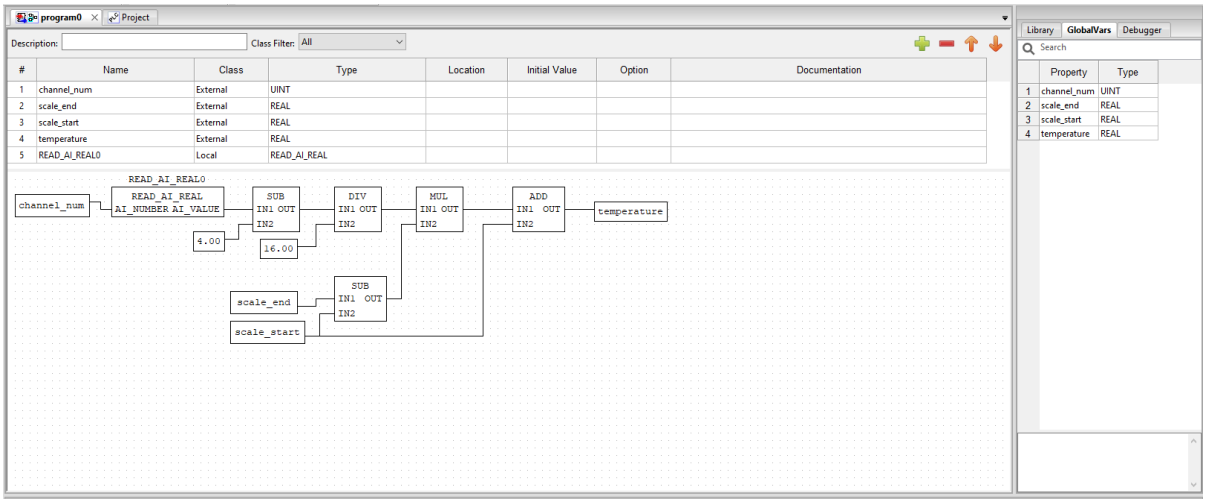


Рис. 43: *Программа на языке FBD*

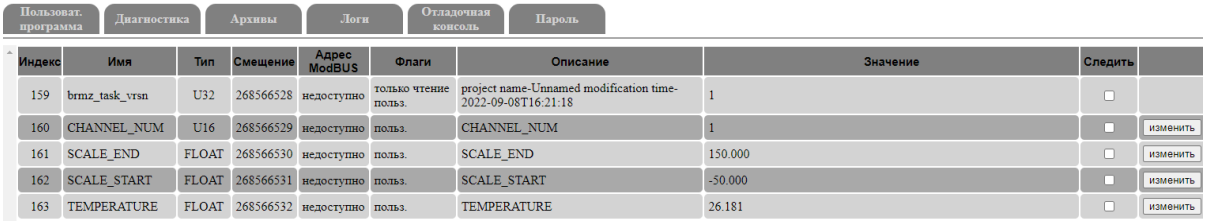


Рис. 44: *WEB-страница ПЛК BRIC*

2.10 AI. Управление каналом DO по заданной уставке

Для данного урока мы преобразуем предыдущую программу с подключением датчика температуры добавив дополнительную программу управления каналами DO. Напишем дополнительную программу на языке ST, которая будет управлять каналами DO_0 и DO_1 в зависимости от температуры.

Примечание: Необходимо добавить дополнительно созданную программу в ресурс, с определением временного интервала

При нагреве чувствительного элемента датчика температуры более 26℃, задействуется дискретный канал DO_0, иначе задействован DO_1.

См.также:

Подробно о аналоговых входах ПЛК BRIC можно узнать по ссылке

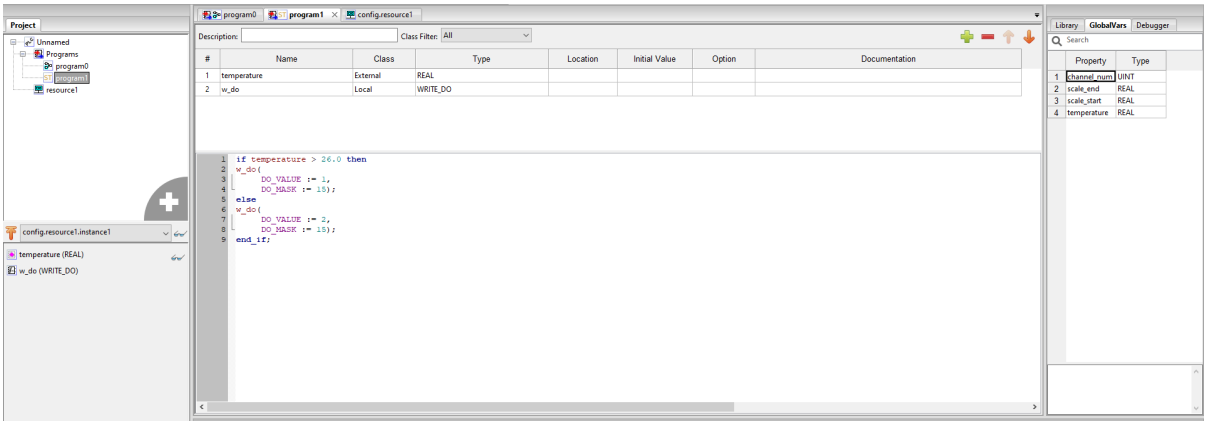


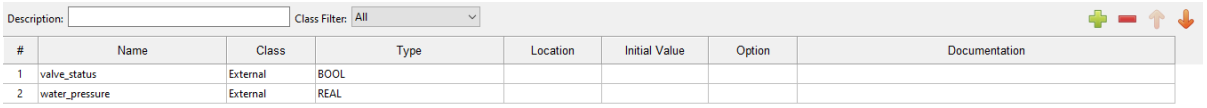
Рис. 45: *Дополнительная программа на языке ST*

Рис. 46: *Управление каналами DO по заданной температуре*

2.11 Тестовая программа. Насосная станция

Напишем программу, эмулирующую простую насосную станцию с гидронакопителем и насосом, управляемым реле давления. Поверхностный насос осуществляет забор воды из открытого источника и подачу её под давлением в гидронакопитель. После выключения насоса, водоснабжение потребителей осуществляется за счет воды, запасенной под давлением в гидронакопителе. После того, как давление воды в гидронакопителе упадет до заданного уровня, реле давления включит насос и цикл повторится.

Для начала напишем программу подачи воды. При открытии вентиля подачи воды в гидронакопителе давление воды должно снижаться, при закрытии – оставаться постоянной. Программу напишем на языке ST.




```
IF valve_status THEN
  (*TRUE - вентиль открыт*)
  IF water_pressure > 0.0 THEN
    (* Уменьшение давления на 3 Па при каждом цикле *)
    water_pressure := water_pressure - 3.0;
  ELSE
    water_pressure := 0.0;
  END_IF;
END_IF;
```

Рис. 47: *Программа valve_pressure на языке ST*

Переменная *water_pressure* – это значение давления воды в гидронакопителе.

Переменна *valve_status* – это статус вентиля/крана подающей воды (0 – вентиль закрыт, 1 – вентиль

открыт).

Напишем первую программу, которая будет имиировать подачу воды после включения насоса. Вторая программа будет управлять самим насосом. Нижним уровнем давления воды в гидронакопителе будем считать значение 60000, верхним – 100000. Включение и выключение насоса осуществляется дискретным каналом DO_0. Программу напишем так же на языке ST. Всю автоматику можно будет включать или выключать переменной *auto_pump_status*.

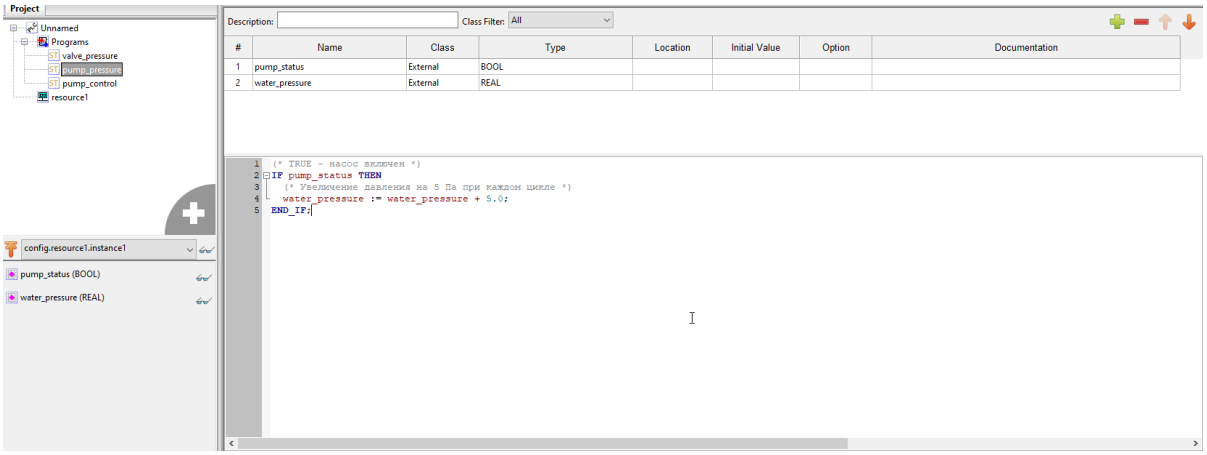


Рис. 48: *Программа pump_pressure на языке ST*

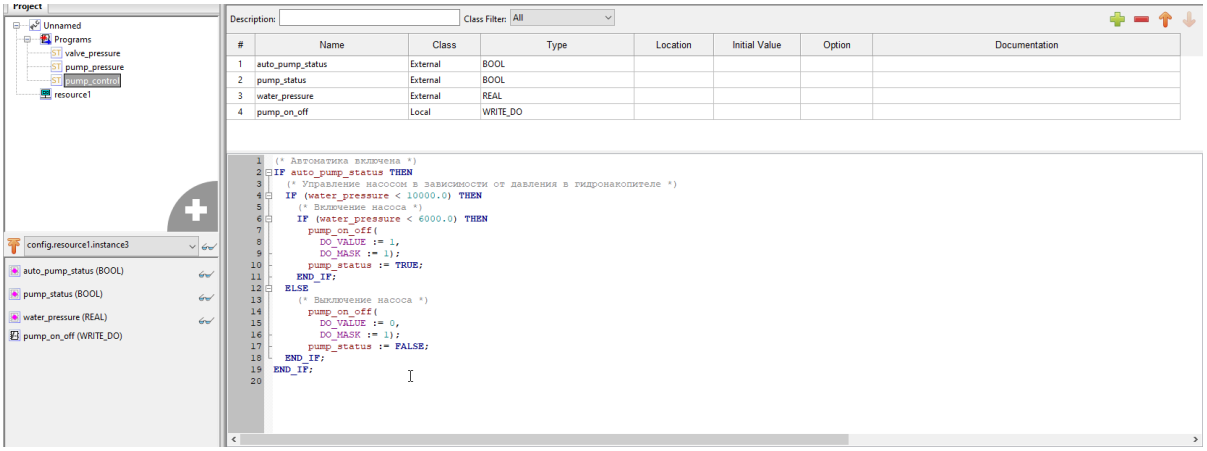


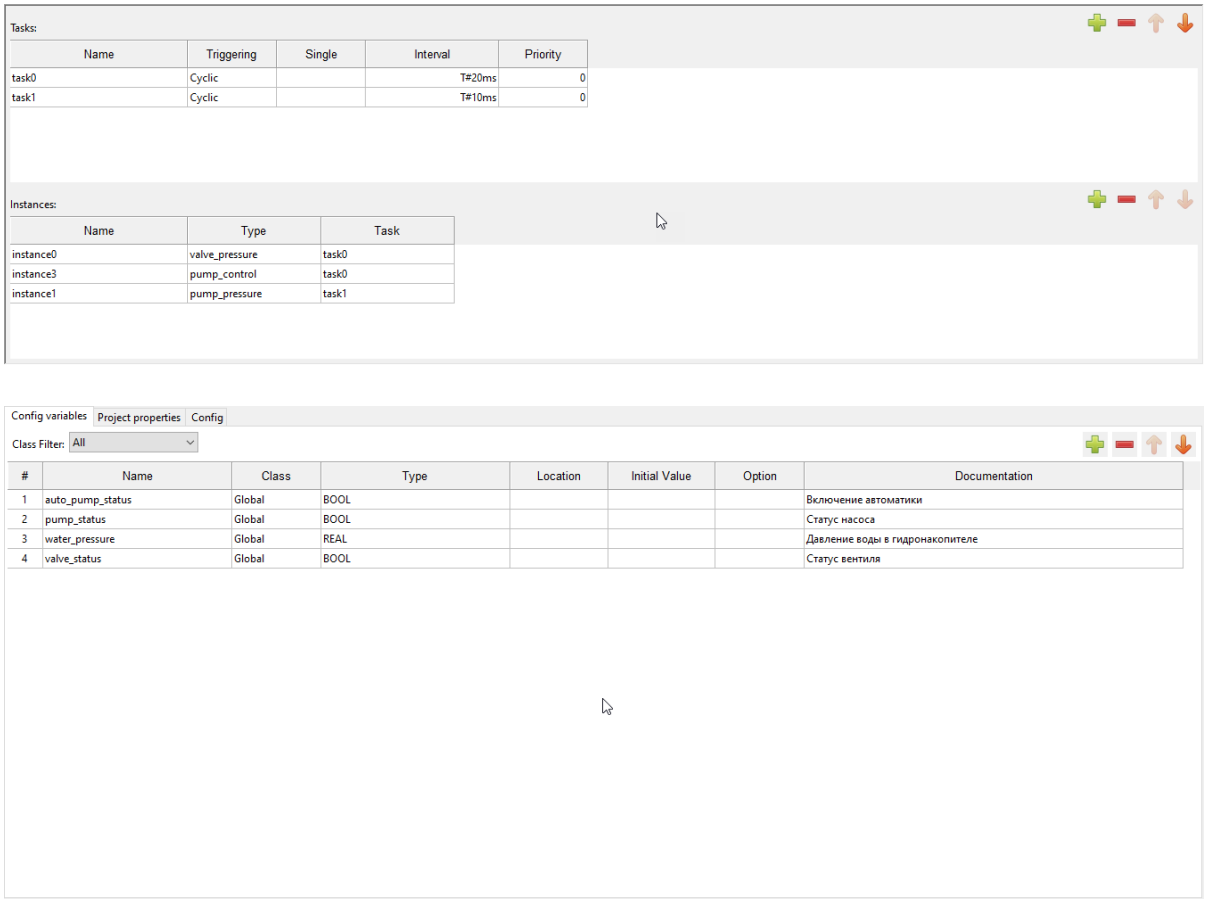
Рис. 49: *Программа pump_control на языке ST*

Переменная *pump_status* – это статус вкл/выкл насоса (0 – насос выключен, 1 – насос включен).

Добавляем задачи на каждую программу соответственно и загружаем в ПЛК.

После загрузки прошивки, заходим в WEB-страницу контроллера. Для включения автоматики переводим *auto_pump_status* в «1» - переменная *pump_status* сразу принимает значение «1», насос включается чтобы заполнить систему и гидронакопитель водой. Включение насоса условно выполняется дискретным выходом DO_0, выключается он при достижении давления в 100000.

После открытия вентиля, давление воды начинает понижаться и при достижении уровня 60000, включается насос и вода в гидронакопителе наполняется до 100000. Если закрыть кран при наполнении воды в гидронакопитель, то насос выключится после достижения высокого уровня. Если выключить автоматику и вентиль оставить открытым, то вода в гидронакопителе полностью опутошится.



Tasks:

Name	Triggering	Single	Interval	Priority
task0	Cyclic		T#20ms	0
task1	Cyclic		T#10ms	0

Instances:

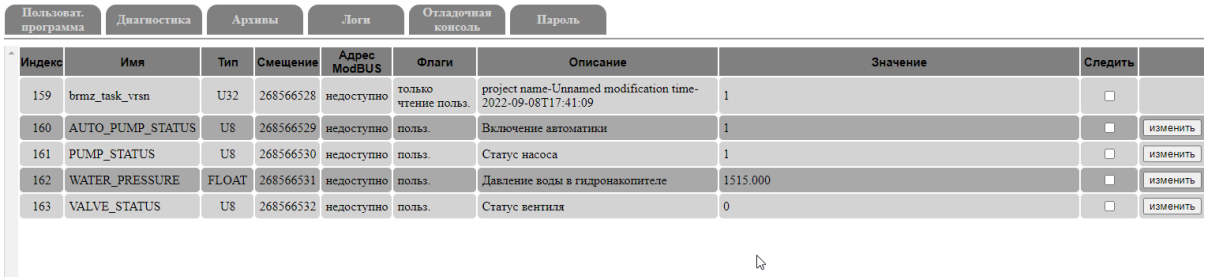
Name	Type	Task
instance0	valve_pressure	task0
instance3	pump_control	task0
instance1	pump_pressure	task1

Config variables | Project properties | Config

Class Filter: All

#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	auto_pump_status	Global	BOOL				Включение автоматики
2	pump_status	Global	BOOL				Статус насоса
3	water_pressure	Global	REAL				Давление воды в гидронакопителе
4	valve_status	Global	BOOL				Статус вентиля

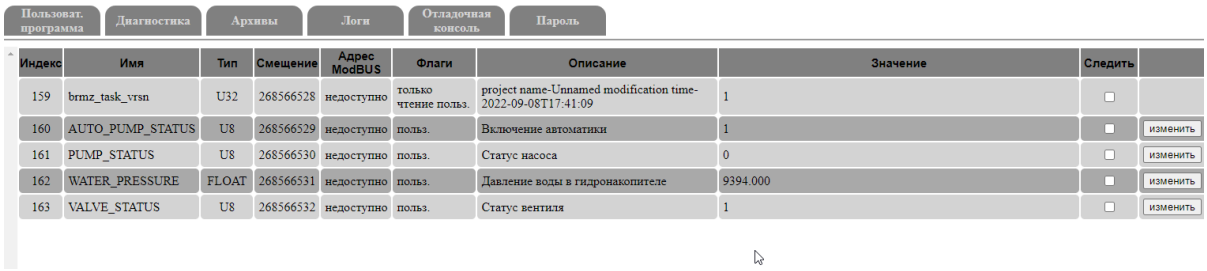
Рис. 50: *Ресурсы и конфигурационные параметры прошивки «Насосная станция»*



Пользоват. программа | Диагностика | Архивы | Логи | Отладочная консоль | Пароль

Индекс	Имя	Тип	Смещение	Адрес ModBUS	Флаги	Описание	Значение	Следить	
159	bmz_task_vrsn	U32	268566528	недоступно	только чтение польз.	project name-Unnamed modification time-2022-09-08T17:41:09	1		
160	AUTO_PUMP_STATUS	U8	268566529	недоступно	польз.	Включение автоматики	1		изменить
161	PUMP_STATUS	U8	268566530	недоступно	польз.	Статус насоса	1		изменить
162	WATER_PRESSURE	FLOAT	268566531	недоступно	польз.	Давление воды в гидронакопителе	1515.000		изменить
163	VALVE_STATUS	U8	268566532	недоступно	польз.	Статус вентиля	0		изменить

Рис. 51: *WEB-страница контроллера. Вентиль открыт*



Пользоват. программа | Диагностика | Архивы | Логи | Отладочная консоль | Пароль

Индекс	Имя	Тип	Смещение	Адрес ModBUS	Флаги	Описание	Значение	Следить	
159	bmz_task_vrsn	U32	268566528	недоступно	только чтение польз.	project name-Unnamed modification time-2022-09-08T17:41:09	1		
160	AUTO_PUMP_STATUS	U8	268566529	недоступно	польз.	Включение автоматики	1		изменить
161	PUMP_STATUS	U8	268566530	недоступно	польз.	Статус насоса	0		изменить
162	WATER_PRESSURE	FLOAT	268566531	недоступно	польз.	Давление воды в гидронакопителе	9394.000		изменить
163	VALVE_STATUS	U8	268566532	недоступно	польз.	Статус вентиля	1		изменить

Рис. 52: *WEB-страница контроллера*

Рис. 53: *Состояние дискретных выходов при включнном насосе*

Глава 3

Модули расширения

3.1 Подключение по межмодульной шине. Подключение по USB. Специальные режимы работы

У ПЛК BRIC имеется несколько модулей расширения, а именно BRIC-AI-16, BRIC-AO-4, BRIC-DI-16 и BRIC-DO-8. Модули расширения подключаются к ПЛК BRIC с использованием топологии «шина» или «кольцо» при помощи межмодульных шлейфов, поставляемых в комплекте с модулями расширения. Для обеспечения стабильного обмена данными между модулями расширения и ПЛК необходимо согласование шлейфа по волновому сопротивлению. С этой целью в каждом модуле расширения и в самом контроллере установленна группа DIP-переключателей «RS-485 («BUS» во второй версии контроллера)», позволяющих подключать к шине терминальные резисторы (терминаторы). В зависимости от используемой топологии соединения, возможны два варианта использования данных переключателей:

- При использовании шинной топологии DIP-переключатели устанавливаются в положение «ON» на устройствах, расположенных по краям шины, не зависимо от того, модуль расширения это или контроллер;
- При использовании топологии типа «Кольцо» устанавить DIP-переключатели в положение «ON» необходимо только на главном контроллер.

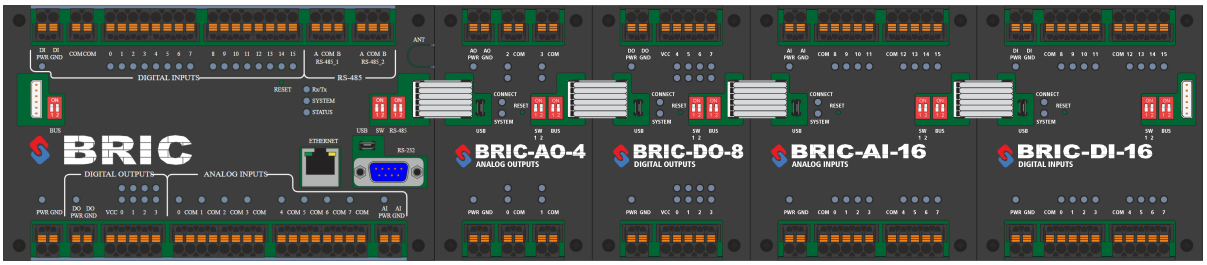


Рис. 1: *Подключение модулей*

Для того, чтобы связь между модулями появилась необходимы следующие условия:

- межмодульные шлейфы подключены
- со стороны неподключенного шлейфа терминаторы межмодульных интерфейсов должны быть подключены с помощью переключателя RS-485/BUS

Важно: Для корректного обмена терминальный резистор межмодульной шины должен быть подключен либо только на главном контроллере, либо на устройствах расположенных по краям межмодульной шины.

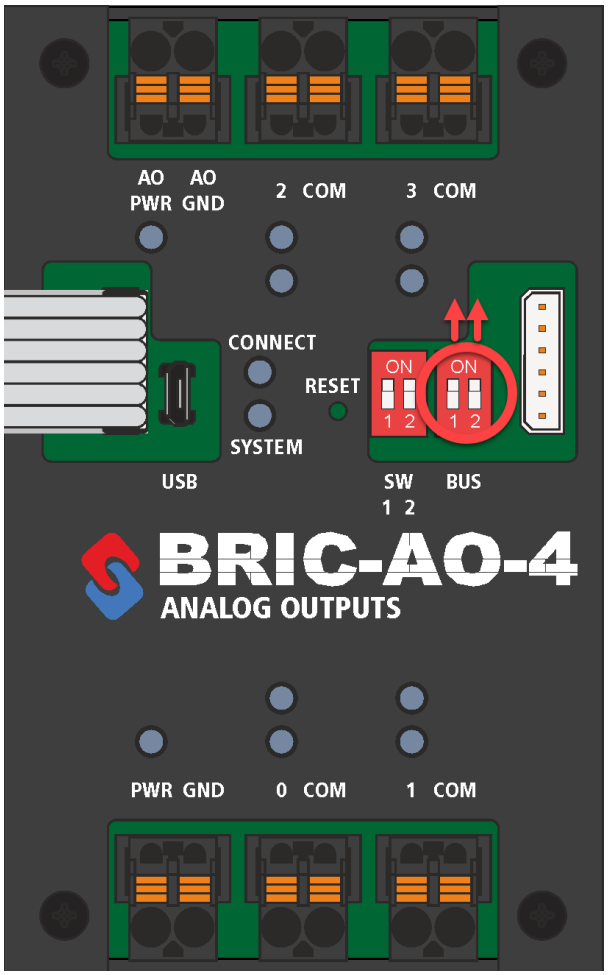


Рис. 2: *Включение терминаторов*

- адреса CANopen у каждого модуля различаются (0-127)

Примечание: При первом подключении модуля расширения ему необходимо присвоить адрес устройства в соответствии с исполняемым пользовательским ПО на master-контроллере. Для этого необходимо подключить модуль по межмодульной шине к master-контроллеру и запитать. Далее в нормальном режиме работы необходимо перевести состояние переключателей в SW-1 > OFF, SW-2 > ON и нажать кнопку RESET. Единовременно на межмодульной CAN-шине может быть только одно устройство в режиме получения нового адреса.

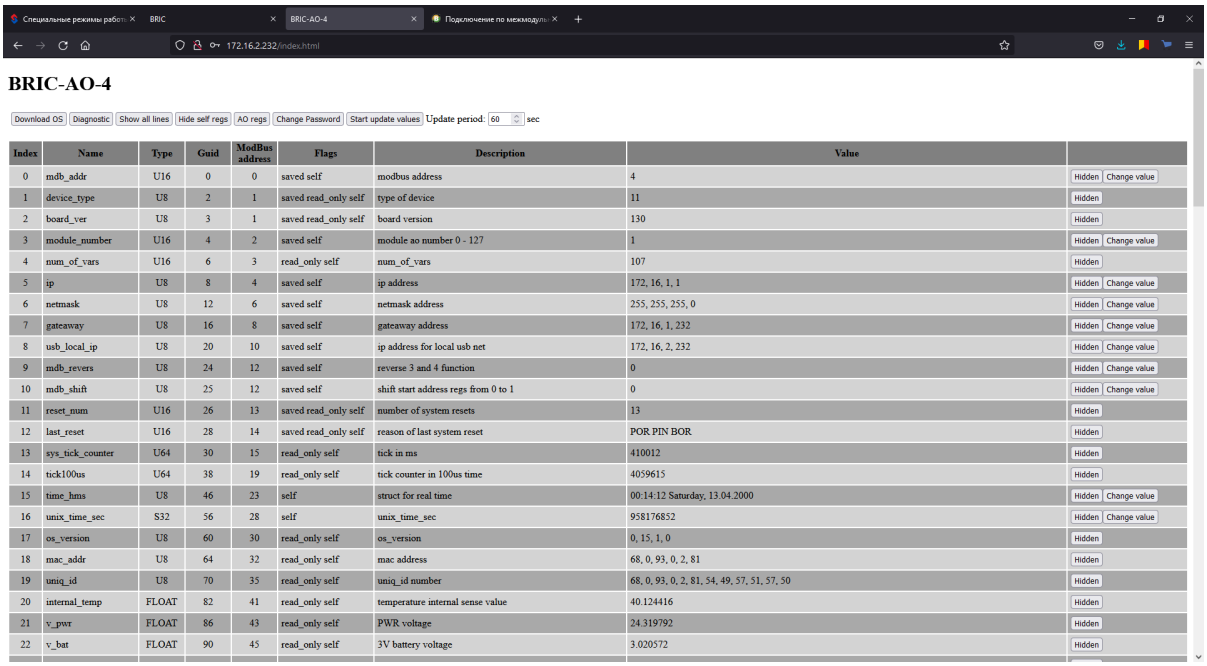
После успешного получения нового адреса светодиод CONNECT загорится оранжевым цветом, что будет свидетельствовать о наличии обмена по CAN-интерфейсу. Возможно, понадобится перезагрузить

главный контроллер.

После успешного присвоения нового адреса необходимо вернуть модуль в нормальный режим работы SW-1 > OFF, SW-2 > OFF.

Рис. 3: *Подключение модуля BRIC-AO-4*

К модулю расширения можно подключиться через интерфейс USB по IP-адресу http://172.16.2.232. Модуль должен быть запитан. После входа в страницу рекомендуется ввести пароль нажав на кнопку «Enter password» и введя «bric» для дальнейшего управления регистрами.



Index	Name	Type	Guid	ModBus address	Flags	Description	Value	
0	mdb_addr	U16	0	0	saved self	modbus address	4	Hidden Change value
1	device_type	U8	2	1	saved read_only self	type of device	11	Hidden
2	board_ver	U8	3	1	saved read_only self	board version	130	Hidden
3	module_number	U16	4	2	saved self	module ao number 0 - 127	1	Hidden Change value
4	num_of_vars	U16	6	3	read_only self	num_of_vars	107	Hidden
5	ip	U8	8	4	saved self	ip address	172, 16, 1, 1	Hidden Change value
6	netmask	U8	12	6	saved self	netmask address	255, 255, 255, 0	Hidden Change value
7	gateaway	U8	16	8	saved self	gateaway address	172, 16, 1, 232	Hidden Change value
8	usb_local_ip	U8	20	10	saved self	ip address for local usb net	172, 16, 2, 232	Hidden Change value
9	mdb_revers	U8	24	12	saved self	reverse 3 and 4 function	0	Hidden Change value
10	mdb_shift	U8	25	12	saved self	shift start address regs from 0 to 1	0	Hidden Change value
11	reset_num	U16	26	13	saved read_only self	number of system resets	13	Hidden
12	last_reset	U16	28	14	saved read_only self	reason of last system reset	POR PIN BOR	Hidden
13	sys_tick_counter	U64	30	15	read_only self	tick in ms	410012	Hidden
14	tick100us	U64	38	19	read_only self	tick counter in 100us time	4059615	Hidden
15	time_hms	U8	46	23	self	struct for real time	00:14:12 Saturday, 13.04.2000	Hidden Change value
16	unix_time_sec	S32	56	28	self	unix_time_sec	958176852	Hidden Change value
17	os_version	U8	60	30	read_only self	os_version	0, 15, 1, 0	Hidden
18	mac_addr	U8	64	32	read_only self	mac address	68, 0, 93, 0, 2, 81	Hidden
19	uniq_id	U8	70	35	read_only self	uniq_id number	68, 0, 93, 0, 2, 81, 54, 49, 57, 51, 57, 50	Hidden
20	internal_temp	FLOAT	82	41	read_only self	temperature internal sense value	40.124416	Hidden
21	v_pwr	FLOAT	86	43	read_only self	PWR voltage	24.319792	Hidden
22	v_bat	FLOAT	90	45	read_only self	3V battery voltage	3.020572	Hidden

Рис. 4: *WEB-страница модуля расширения*

В данной WEB-странице также имеется возможность установки номера модуля. Для этого необходимо нажать на кнопку «Change value» в строке «module_number», ввести нужное значение и нажать на кнопку «Set new reg value».

3.2 BRIC-AO-4. Работа с регистрами PDO

Для полного понимания данного урока необходимо обратиться к терминологии. Протокол CANopen определяет несколько методов передачи сообщений по сети CAN. Эти сообщения называются объектами связи (communication objects). Есть 2 базовых, отличающихся друг от друга способа передачи данных:

- **Service Data Objects (SDO)** механизм обычно используется для конфигурирования устройств, достоинство это то, что имеется возможность контролировать соединение и ответ устройства. Недостатки - более низкий приоритет по сравнению с PDO.
- **Process Data Objects (PDO)** механизм используется для передачи с высокой скоростью высокоприоритетных данных, так как PDO сообщения не содержат никаких дополнительных про-

токольных данных. Обычно используется для заранее сконфигурированного регулярного обмена информацией.

Коротко говоря, в Beremize SDO регистры опрашиваются ведущим ПЛК при указании пользователем, а PDO - регистры опрашиваются автоматически.

Напишем программу AO_PDO на языке FBD. Добавим модуль расширения AO. Для этого нажмем на область «Project - > Modules support». Нажимаем правой кнопкой на появившийся «module_0 -> Add AO».

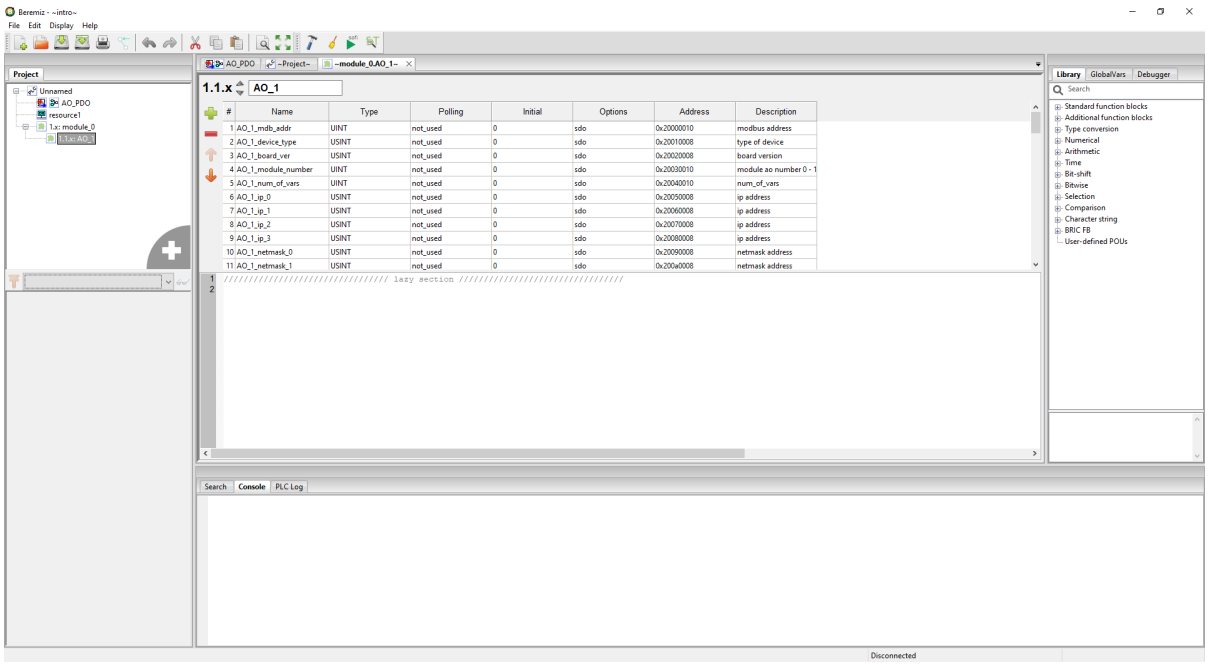


Рис. 5: *Добавление AO в Beremiz*

Модуль расширения BRIC-AO-4 имеет 4 регистра PDO. В окне Polling записана «1», а в Options «pdor_0x200».

AO_ao_val_x - это выходное значение аналогового канала в единицах АЦП, с диапазоном 0-4095.

Наша программа будет управлять аналоговым каналом AO_0. Из окна настроек модуля расширения мы будем использовать регистр PDO AO_1_ao_val_0. Добавим его в программу с корректным типом данных.

Задаем значение, например, 2000 в «ao_val» и увидим результат в действии.

Примечание: Подробно о модуле расширения BRIC-AO-4 можно узнать по ссылке

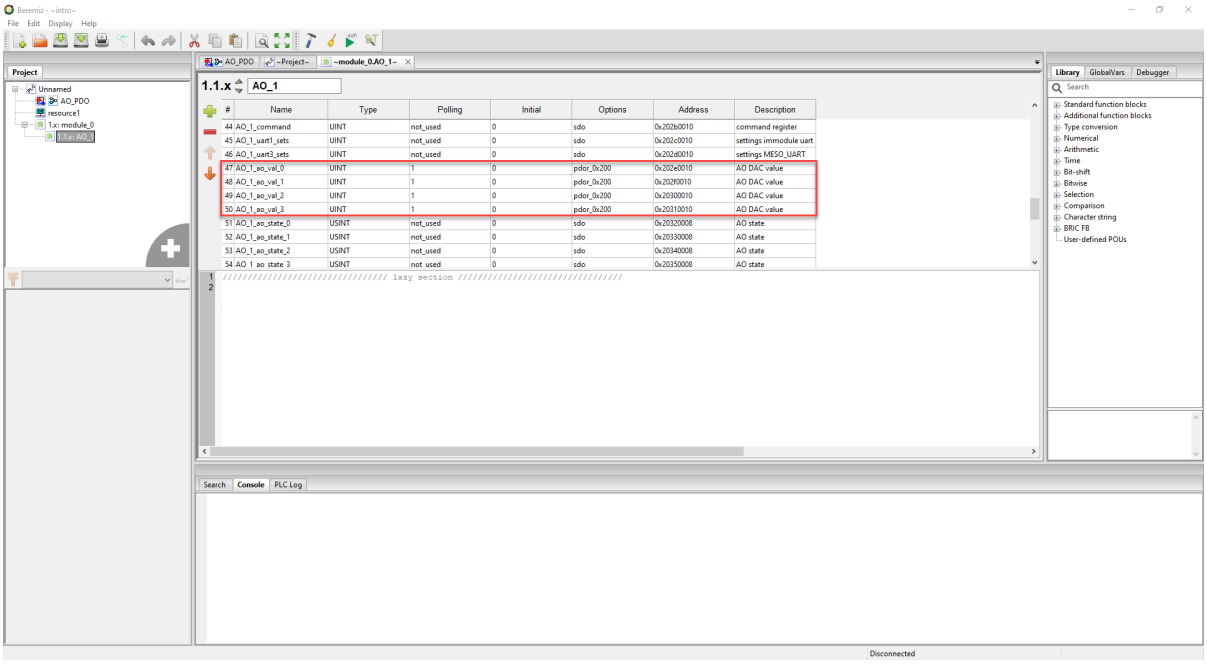


Рис. 6: *PDO регистры в AO*

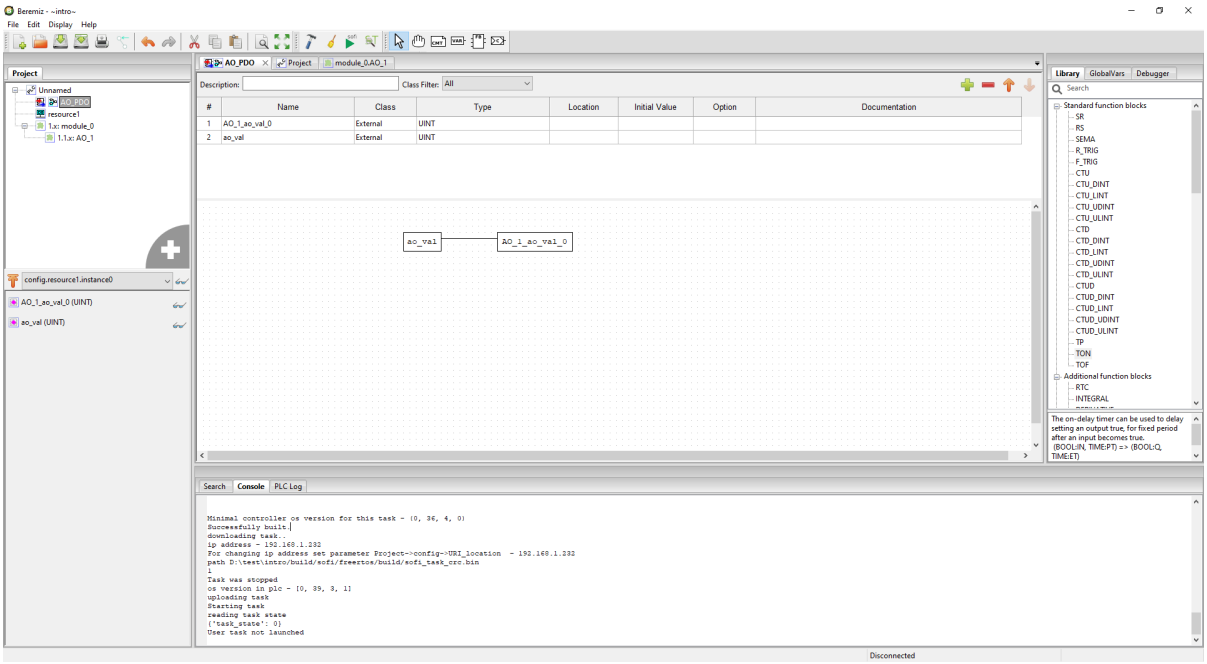


Рис. 7: *Программа AO_PDO*

Пользоват. программа | Диагностика | Архивы | Логи | Отладочная консоль | Пароль

Индекс	Имя	Тип	Смещение	Адрес ModBUS	Флаги	Описание	Значение	Следить	
159	brmz_task_vrsn	U32	268566528	недоступно	только чтение польз.	project name-Unnamed modification time-2022-09-09T10:32:54	1	☐	
160	AO_VAL	U16	268566529	недоступно	польз.	AO_VAL	2000	☐	изменить

Рис. 8: *WEB-страница программы*

Рис. 9: *Программа в действии*

3.3 BRIC-DI-16. Работа с регистрами PDO

В данном уроке разберем регистр PDO модуля расширения BRIC-DI-16. За логическое состояние дискретных входов модуля отвечает параметр DI_state. У данного параметра диапазон от 0 до 65535, каждый бит содержит состояние отдельного канала.

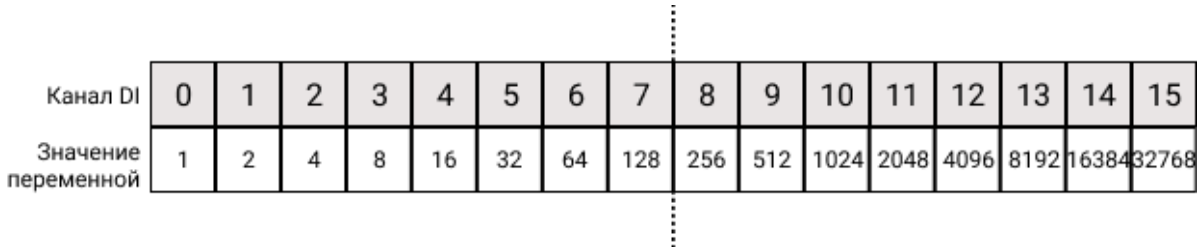


Рис. 10: *Определение логического состояния канала*

Подключим модуль расширения к ПЛК.

Рис. 11: *ПЛК с модулем расширения*

Подключим дискретные выходы ПЛК к дискретным входам модуля расширения в соответствии со схемой, приведенной ниже.

Напишем программу на языке ST, в которой состояние дискретных выходов ПЛК будет зависеть от состояния дискретных входов модуля расширения. Добавим PDO-регистр DI_1_di_state в основную программу. Состояние дискретных входов будет записываться в переменную DI_STATE.Добавим PDO-регистр *DI_di_state* в основную программу. Состояние дискретных входов будет записываться в переменную *DI_STATE*.

Конечный вариант программы представлен на рисунке ниже.

Примечание: Подробно о модуле расширения BRIC-DI-16 можно узнать по ссылке

3.4 BRIC-DO-8. Работа с регистрами PDO

Напишем программу для работы с регистрами PDO модуля расширения BRIC-DO-8. У данного модуля всего 2 регистра данного типа: DO_do_sc_ctrl и DO_do_ctrl.

DO_do_sc_ctrl - регистр, отвечающий за срабатывание программной защиты от короткого замыкания.

DO_do_ctrl - регистр, отвечающий за управление логическим состоянием дискретных каналов.

Программа будет простой и написана на языке FBD. Добавляем модуль DO в Beremiz.

В основной программе создаем внешние переменные для передачи вводимых значений PDO-регистрам модуля расширения. Для этого создаем внешние переменные с именами PDO-регистров, через которые наши данные передаются в соответствующие регистры модуля расширения. С этой целью, при

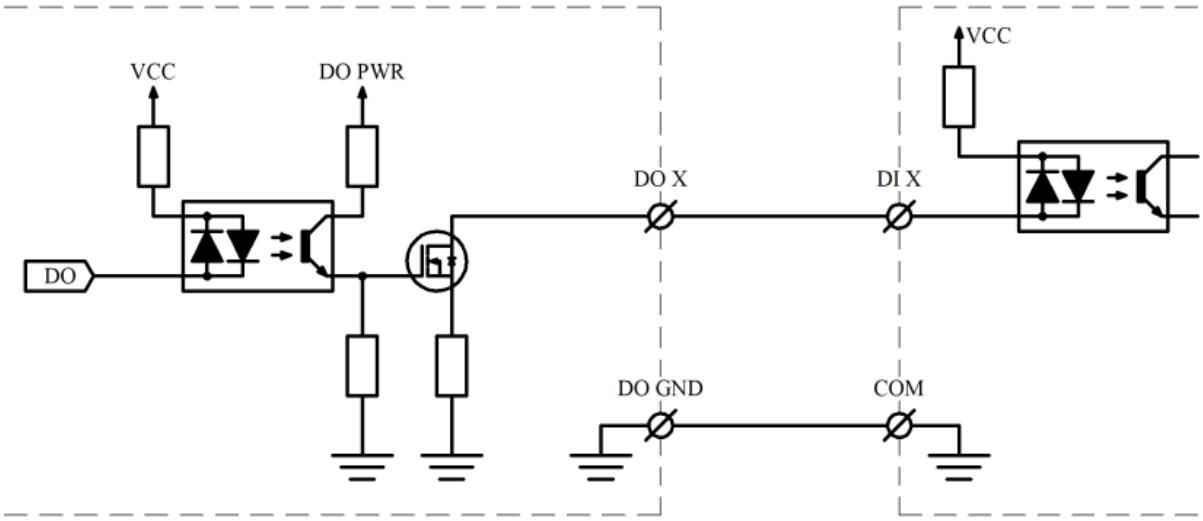


Рис. 12: *Схема подключения*

Description: Class Filter: All

#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	DI_1_di_state	External	UDINT				
2	DI_STATE	External	UDINT				
3	W_DO	Local	WRITE_DO				

```
W_DO(
    DO_VALUE := 1,
    DO_MASK := 15);
DI_STATE := DI_1_di_state;

IF (DI_STATE = 1) THEN
   W_DO(
    DO_VALUE := 2,
    DO_MASK := 15);
END_IF;
IF (DI_STATE = 2) THEN
   W_DO(
    DO_VALUE := 4,
    DO_MASK := 15);
END_IF;
IF (DI_STATE = 4) THEN
   W_DO(
    DO_VALUE := 8,
    DO_MASK := 15);
END_IF;
IF (DI_STATE = 8) THEN
   W_DO(
    DO_VALUE := 1,
    DO_MASK := 15);
END_IF;
```

Рис. 13: *Программа для проверки PDO-регистра модуля BRIC-DI-16*

Рис. 14: *Результат программы*

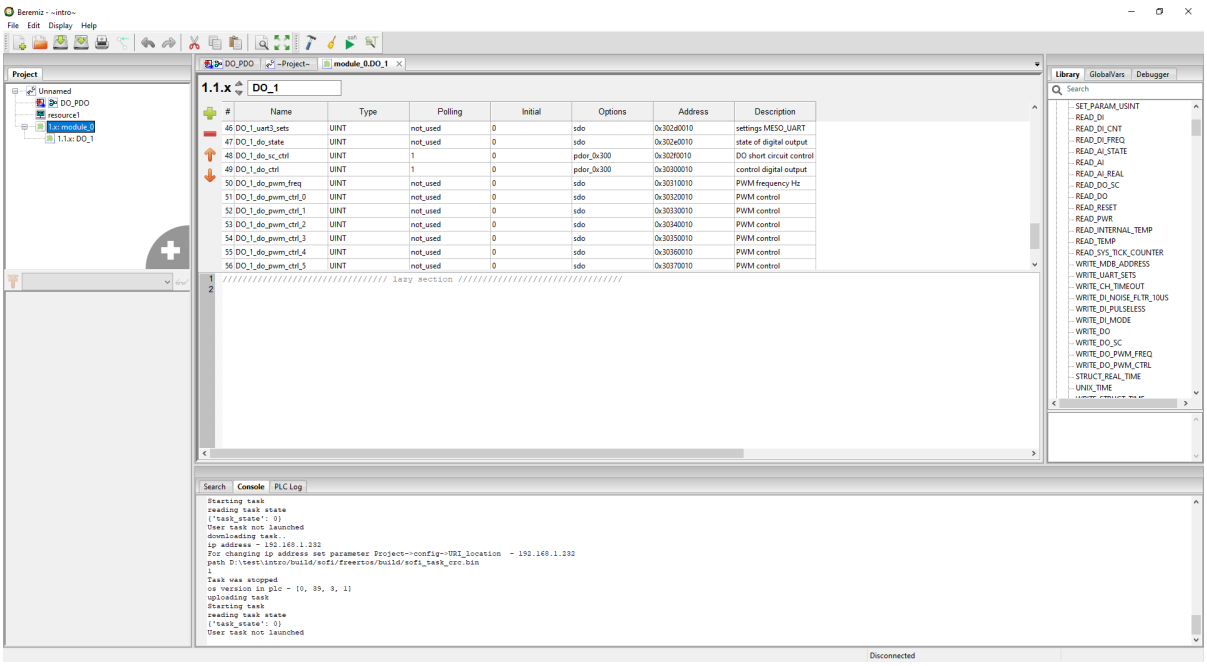


Рис. 15: *Добавление модуля DO*

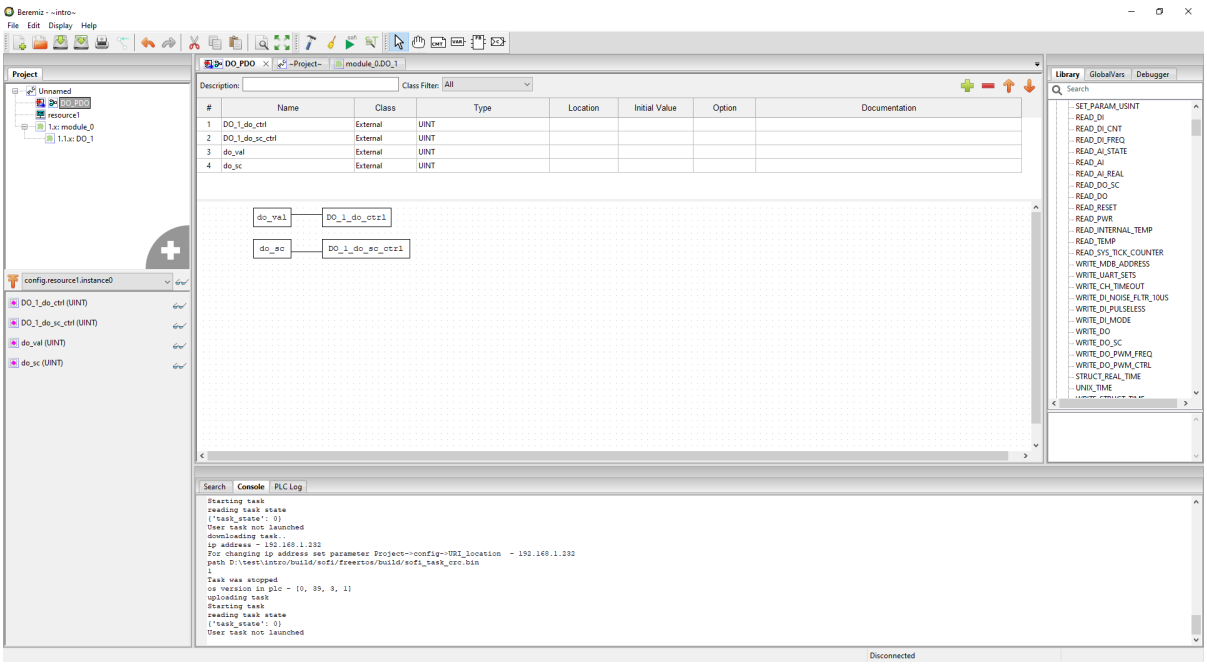


Рис. 16: *Программа DO_PDO*

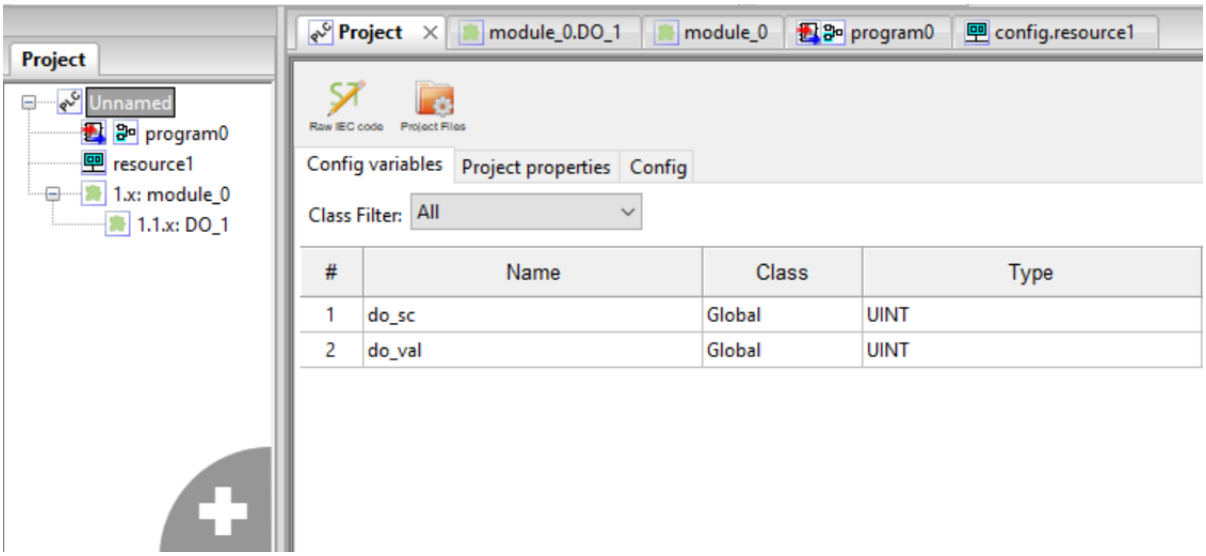


Рис. 17: *Глобальные переменные*

добавлении модуля в ПЛК BRIC, Beremiz создает Интерфейс обмена с модулями по шине CAN (1.x: module_0) и Словарь объекта (OD) для нашего модуля расширения (1.1.x: DO_1).

Запускаем WEB-страницу контроллера, и задаем велечину *do_val* равной, например, 13.

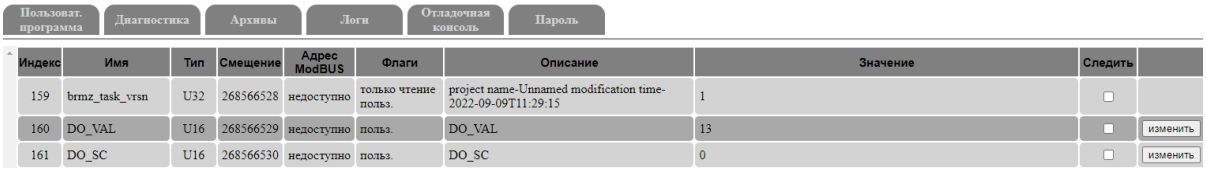


Рис. 18: *WEB-страница контроллера*

После задания значений включаются каналы DO_0, DO-2 и DO_3 модуля расширения BRIC-DO-8. В данном случае маска разрешения изменения состояния дискретных выходов в ПЛК BRIC не применяется (состояние выходов модулей расширения управляется «напрямую» через PDO-регистры по шине CAN.

Канал DO	0	1	2	3	4	5	6	7
Значение переменной	1	2	4	8	16	32	64	128
Бит	1	0	1	1	0	0	0	0

Рис. 19: *Запись состояния дискретного выхода при do_val = 13*

В переменную *do_sc* можно задать значение от 0 до 255, в зависимости какие каналы необходимо программно защитить от короткого замыкания. Например, запишем значение, равную в 32.

Состояние выходов не поменяется, но записывается программная защита для канала DO_5. Убедиться в этом можно через WEB-страницу модуля расширения. Нажимая кнопку «Hide self regs», а далее кноп-

Рис. 20: *Результат программы*

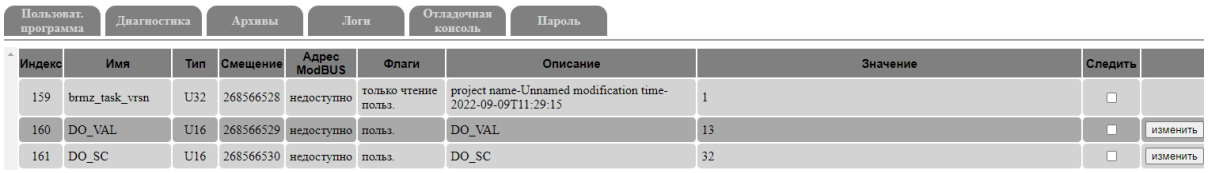


Рис. 21: *WEB-страница контроллера*

ку «DO regs» можно увидеть указанную стрелкой бит программной защиты от КЗ, соответствующий каналу DO_5.

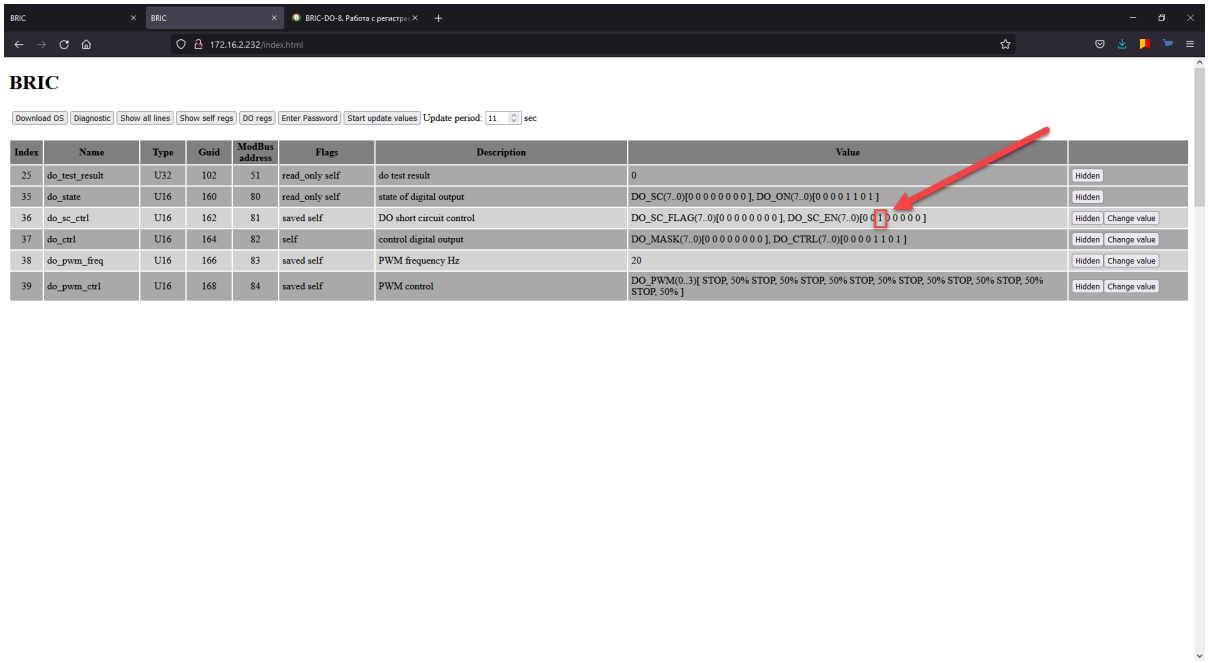


Рис. 22: *WEB-страница модуля*

Примечание: Подробно о модуле расширения BRIC-DO-8 можно узнать по ссылке

3.5 BRIC-AI-16. Работа с регистрами PDO

У модуля расширения BRIC-AI-16 имеется больше всего PDO-регистров. В данном уроке напишем программу на языке FBD, подключим так же модуль расширения BRIC-AO-4 для наглядности примера.

Примечание: Про подключение нескольких модулей и назначение адресов можно изучить по ссылке.

Наша программа будет обрабатывать входные аналоговые сигналы. Добавим модули расширения в программу.

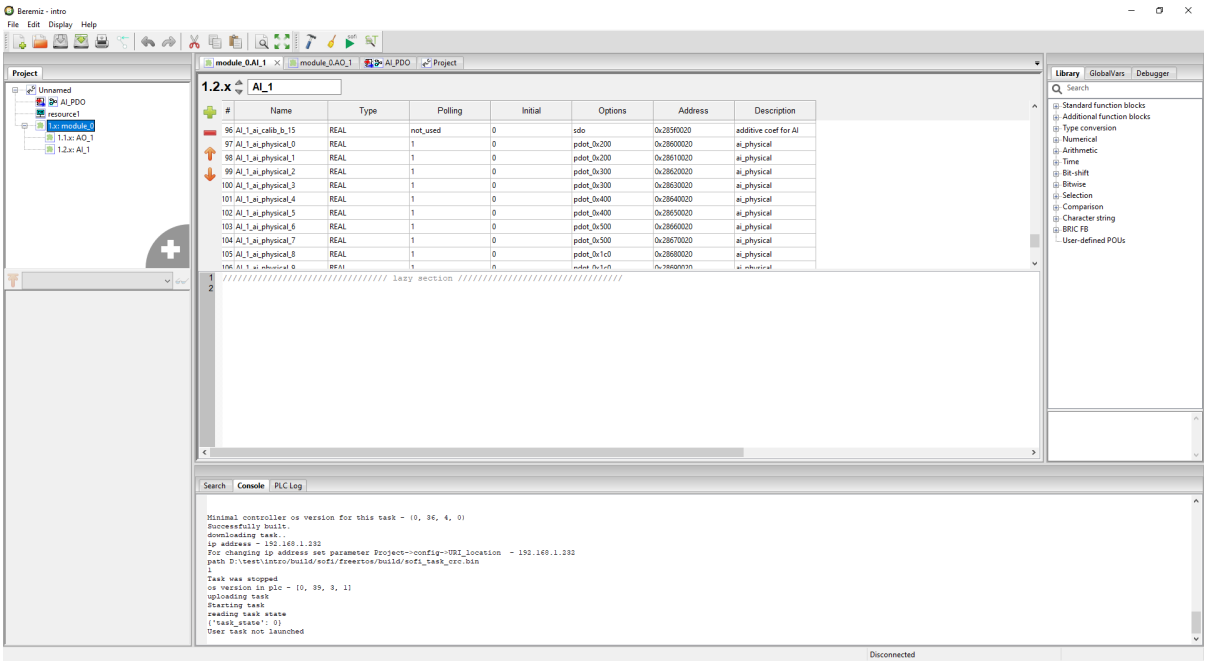


Рис. 23: *Добавленные модули в Beremiz*

Модули расширения добавлены, соединены и адреса настроены. Терминальный резистор (BUS) подключен на главном контроллере.

Рис. 24: *Добавленные модули к ПЛК*

Создадим переменные, связанные с PDO-регистрами модуля AO и запишим в них выходные значения аналоговых каналов в единицах ЦАП из диапазоном 0-4095

Далее добавляем PDO-регистры AI_1_ai_physical с типом данных REAL, тем самым мы узнаем какая сила тока в mA протекает в первых 4-х каналах. Данные будут записываться в переменные *ai_real*.

Загружаем программу в ПЛК. Изначально соединяем каналы AO с каналами AI.

В WEB-странице выводятся данные о силе тока каждого задействованного канала.

Проверим достоверность данных, например, нулевого канала AO_0. Как видим, результаты совпадают и данные достоверны.

Примечание: Подробно о модуле расширения BRIC-AI-16 можно узнать по тут

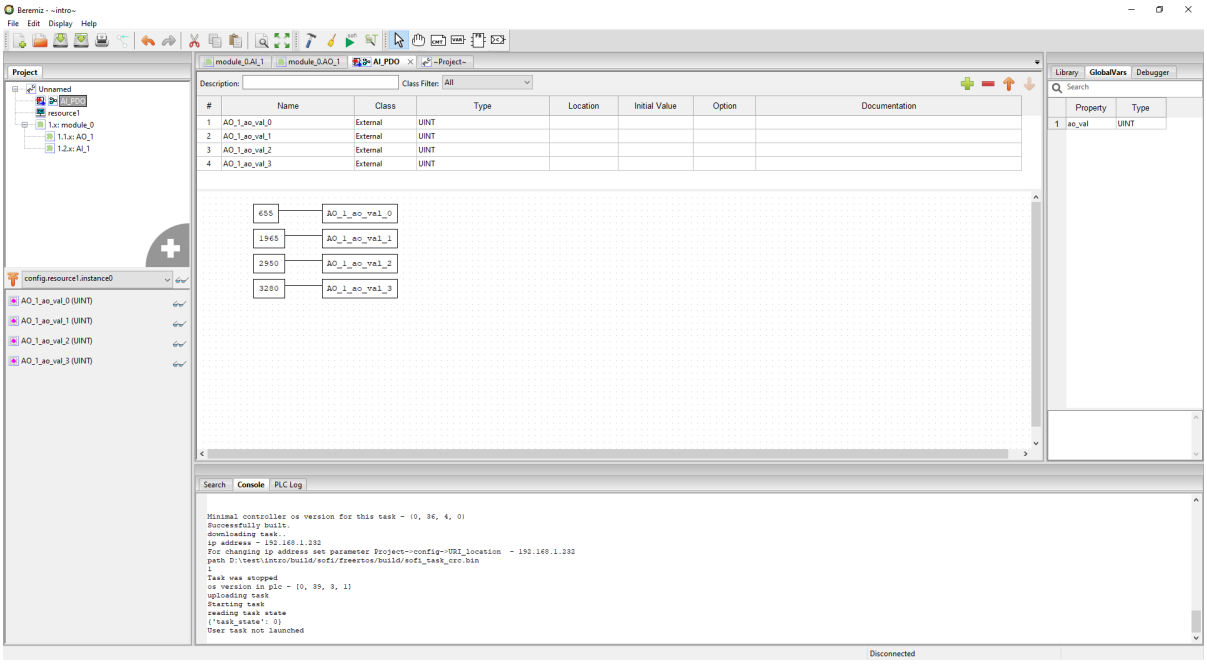


Рис. 25: *Работа с регистрами pdo модуля AO*

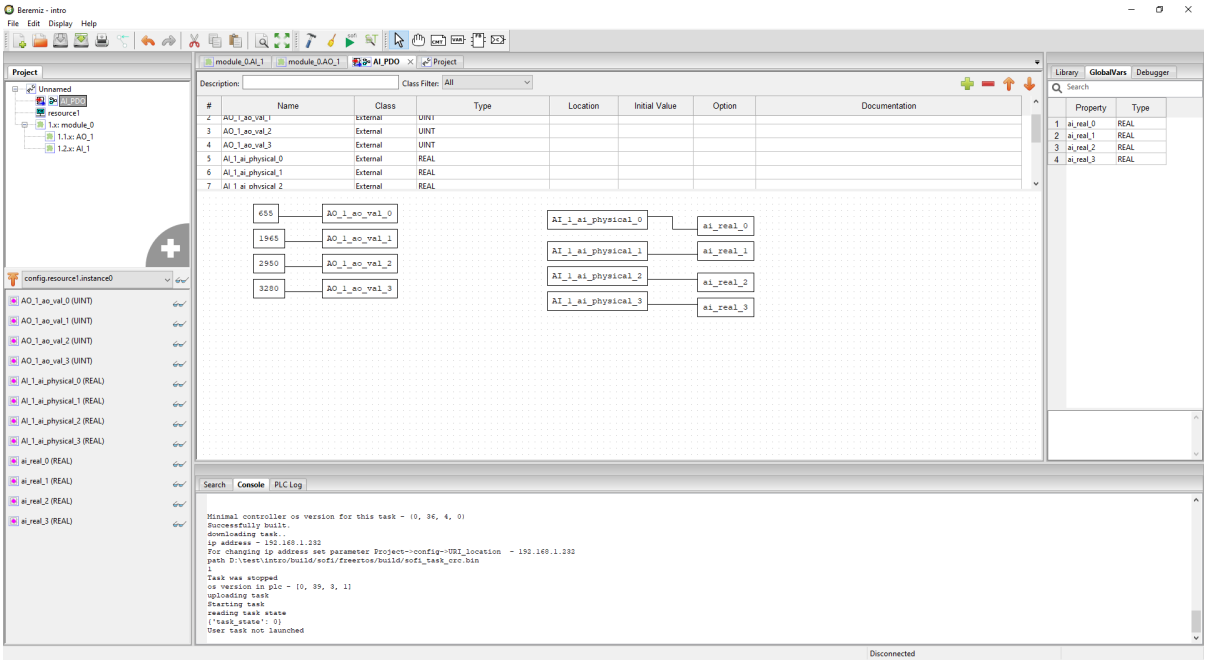
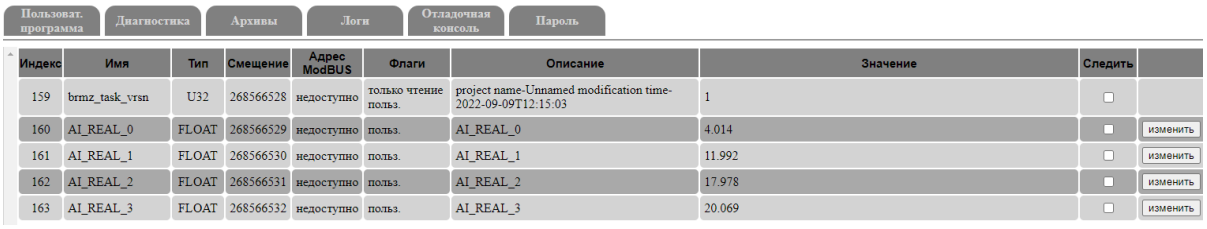


Рис. 26: *Программа AI_PDO*

Рис. 27: *Результат программы*



Индекс	Имя	Тип	Смещение	Адрес ModBUS	Флаги	Описание	Значение	Следить	
159	brmz_task_vrsn	U32	268566528	недоступно	только чтение польз.	project name-Unnamed modification time-2022-09-09T12:15:03	1	☐	
160	AI_REAL_0	FLOAT	268566529	недоступно	польз.	AI_REAL_0	4.014	☐	изменить
161	AI_REAL_1	FLOAT	268566530	недоступно	польз.	AI_REAL_1	11.992	☐	изменить
162	AI_REAL_2	FLOAT	268566531	недоступно	польз.	AI_REAL_2	17.978	☐	изменить
163	AI_REAL_3	FLOAT	268566532	недоступно	польз.	AI_REAL_3	20.069	☐	изменить

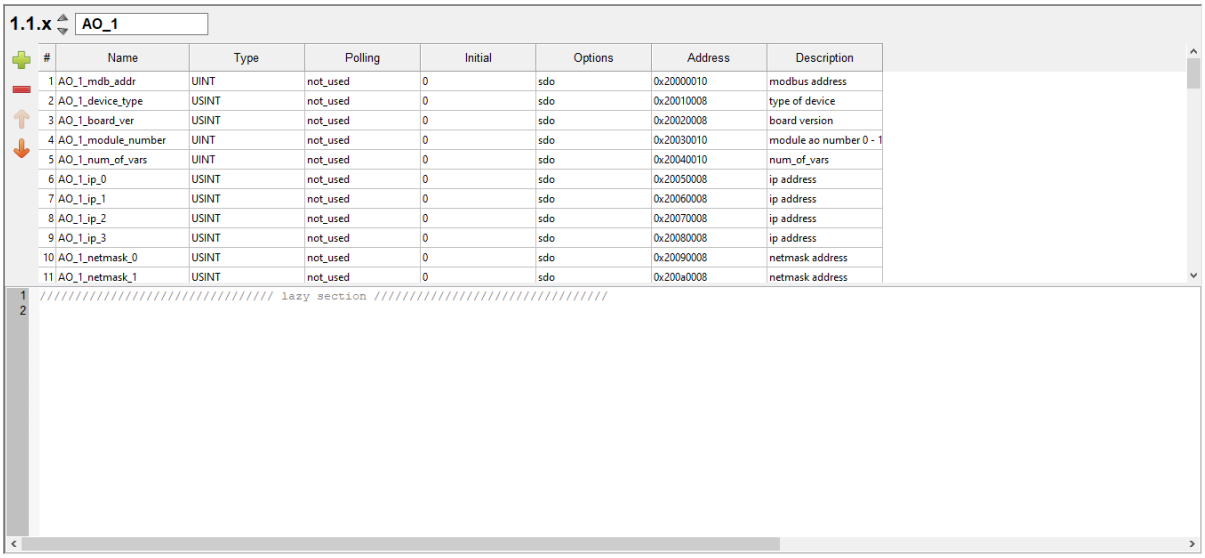
Рис. 28: *WEB-страница программы*

Рис. 29: *Проверка канала AO_0*

3.6 Добавление и работа с регистрами SDO. Пример на модуле BRIC-AO-4.

Примечание: Данный урок показан на примере модуля расширения BRIC-AO-4. Аналогичным образом добавляются и используются регистры SDO и для остальных модулей.

Для проверки регистров SDO модуля расширения BRIC-AO-4 напишем программу на языке FBD. Добавляем модуль AO в Beremiz, и откроем окно настроек. Все регистры, кроме AO_ao_val_x являются регистрами SDO.



#	Name	Type	Polling	Initial	Options	Address	Description
1	AO_1_mdb_addr	UINT	not_used	0	sdo	0x20000010	modbus address
2	AO_1_device_type	USINT	not_used	0	sdo	0x20010008	type of device
3	AO_1_board_ver	USINT	not_used	0	sdo	0x20020008	board version
4	AO_1_module_number	UINT	not_used	0	sdo	0x20030010	module ao number 0 - 1
5	AO_1_num_of_vars	UINT	not_used	0	sdo	0x20040010	num_of_vars
6	AO_1_ip_0	USINT	not_used	0	sdo	0x20050008	ip address
7	AO_1_ip_1	USINT	not_used	0	sdo	0x20060008	ip address
8	AO_1_ip_2	USINT	not_used	0	sdo	0x20070008	ip address
9	AO_1_ip_3	USINT	not_used	0	sdo	0x20080008	ip address
10	AO_1_netmask_0	USINT	not_used	0	sdo	0x20090008	netmask address
11	AO_1_netmask_1	USINT	not_used	0	sdo	0x200a0008	netmask address

Рис. 30: *SDO регистры в AO*

Наша программа будет записывать новый адрес modbus, а также читать CANopen-адрес модуля расширения. В окне регистров модуля расширения находим SDO-регистры AO_1_mdb_addr и AO_1_module_number. В столбце Polling для первого регистра прописываем «write», а для второго «read».

Добавим регистры в основную программу.

Загружаем программу в ПЛК и заходим на WEB-страницу контроллера через Ethernet. Также зайдем на WEB-страницу модуля расширения через USB-порт.

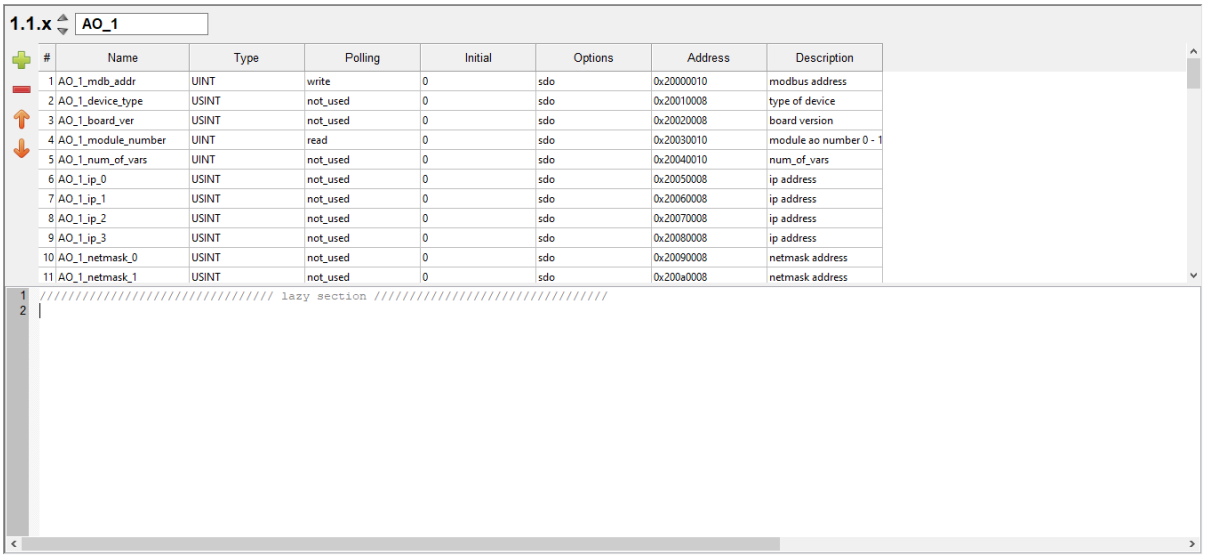


Рис. 31: *Запись в Polling SDO регистров в AO*

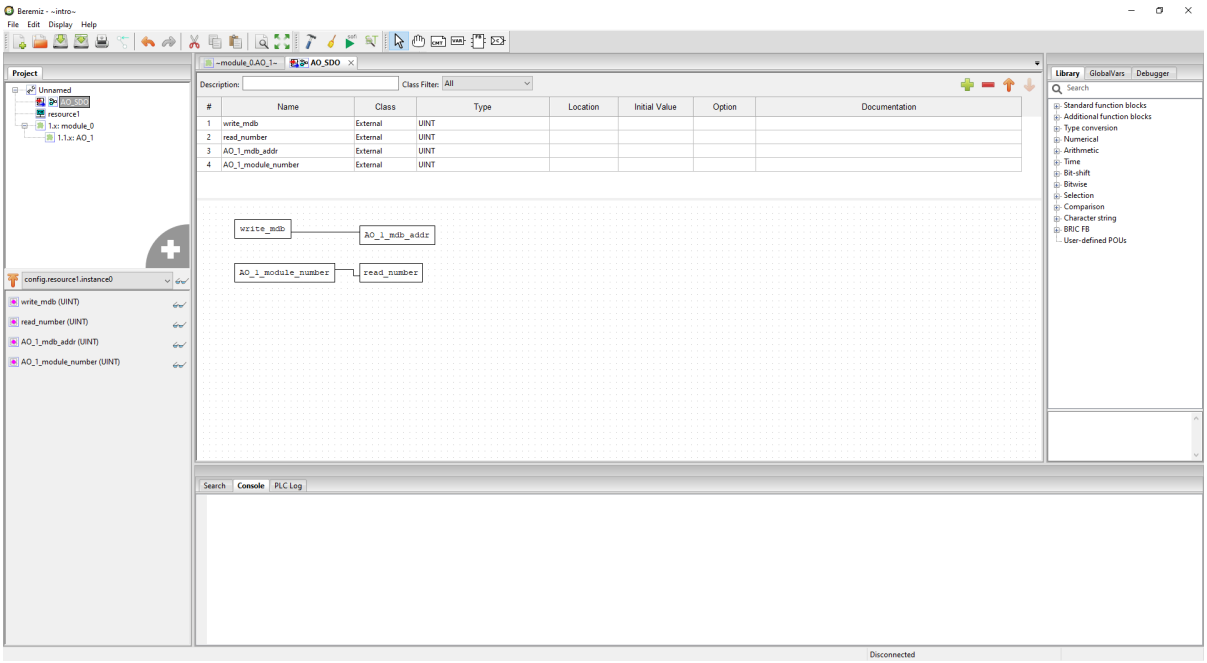


Рис. 32: *Программа для регистров SDO*

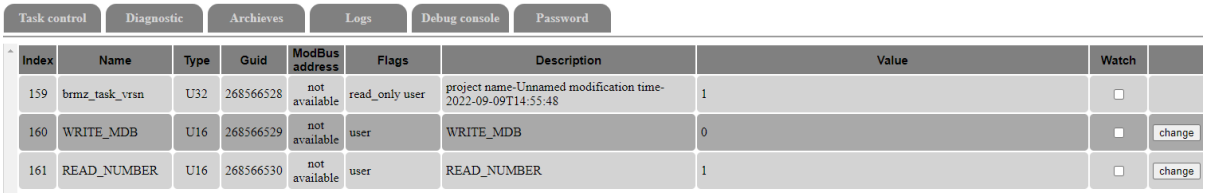
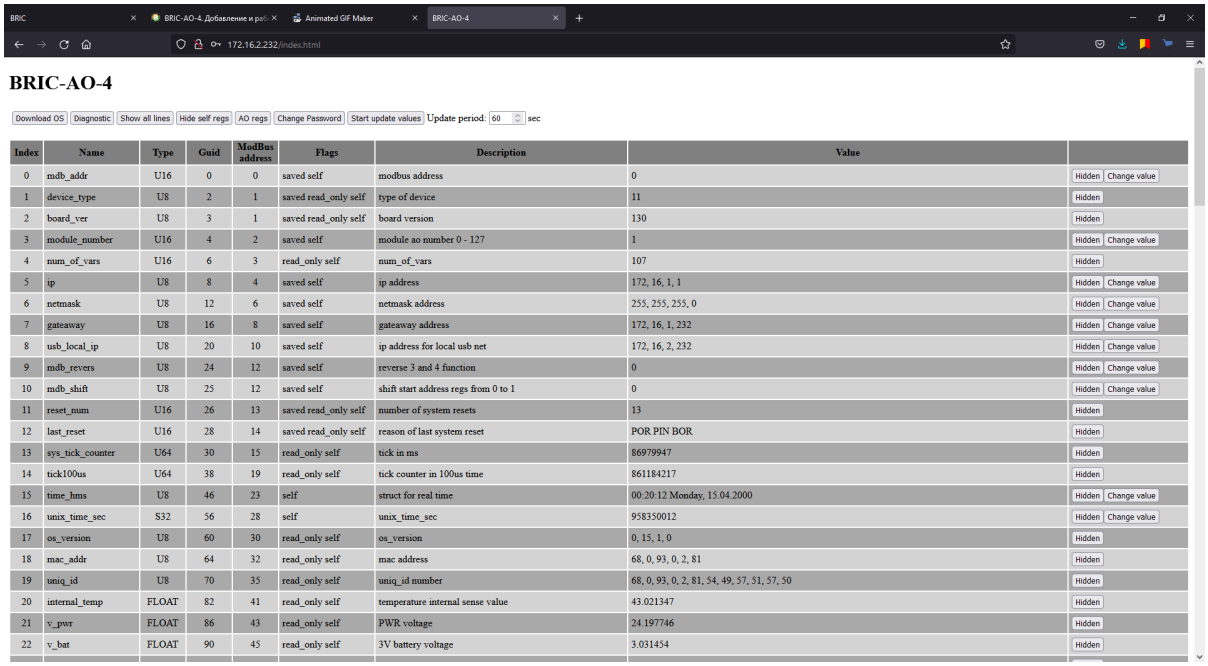


Рис. 33: *WEB-страница ПЛК*



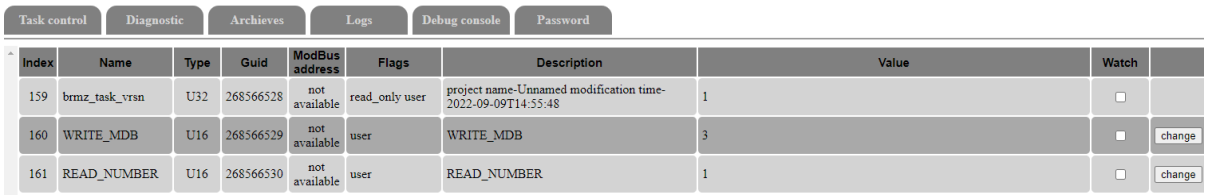
BRIC-AO-4

Download OS | Diagnostic | Show all lines | Hide self regs | AO regs | Change Password | Start update values | Update period: 60 sec

Index	Name	Type	Guid	ModBus address	Flags	Description	Value	
0	mdb_addr	U16	0	0	saved self	modbus address	0	Hidden Change value
1	device_type	U8	2	1	saved read_only self	type of device	11	Hidden
2	board_ver	U8	3	1	saved read_only self	board version	130	Hidden
3	module_number	U16	4	2	saved self	module ao number 0 - 127	1	Hidden Change value
4	num_of_vars	U16	6	3	read_only self	num_of_vars	107	Hidden
5	ip	U8	8	4	saved self	ip address	172, 16, 1, 1	Hidden Change value
6	netmask	U8	12	6	saved self	netmask address	255, 255, 255, 0	Hidden Change value
7	gateaway	U8	16	8	saved self	gateaway address	172, 16, 1, 232	Hidden Change value
8	usb_local_ip	U8	20	10	saved self	ip address for local usb net	172, 16, 2, 232	Hidden Change value
9	mdb_revers	U8	24	12	saved self	reverse 3 and 4 function	0	Hidden Change value
10	mdb_shift	U8	25	12	saved self	shift start address regs from 0 to 1	0	Hidden Change value
11	reset_num	U16	26	13	saved read_only self	number of system resets	13	Hidden
12	last_reset	U16	28	14	saved read_only self	reason of last system reset	POR PIN BOR	Hidden
13	sys_tick_counter	U64	30	15	read_only self	tick in ms	86979947	Hidden
14	tick100us	U64	38	19	read_only self	tick counter in 100us time	861184217	Hidden
15	time_hms	U8	46	23	self	struct for real time	00:20:12 Monday, 15.04.2000	Hidden Change value
16	unix_time_sec	S32	56	28	self	unix_time_sec	958350012	Hidden Change value
17	os_version	U8	60	30	read_only self	os_version	0, 15, 1, 0	Hidden
18	mac_addr	U8	64	32	read_only self	mac address	68, 0, 93, 0, 2, 81	Hidden
19	uniq_id	U8	70	35	read_only self	uniq_id number	68, 0, 93, 0, 2, 81, 54, 49, 57, 51, 57, 50	Hidden
20	internal_temp	FLOAT	82	41	read_only self	temperature internal sense value	43.021347	Hidden
21	v_pwr	FLOAT	86	43	read_only self	PWR voltage	24.197746	Hidden
22	v_bat	FLOAT	90	45	read_only self	3V battery voltage	3.031454	Hidden

Рис. 34: *WEB-страница модуля расширения*

Как видно на странице модуля, modbus-адрес поменялся на «0», так как по умолчанию «write_mdb» инициализируется нулевым значением. Также на этой странице можно увидеть, что «module_number» равно 1, как и полученное значение переменной «read_number». При изменении значения переменной «write_mdb», например на «3», изменится и «mdb_addr» модуля.



Task control | Diagnostic | Archieves | Logs | Debug console | Password

Index	Name	Type	Guid	ModBus address	Flags	Description	Value	Watch	
159	brmz_task_vrsn	U32	268566528	not available	read_only user	project name-Unnamed modification time-2022-09-09T14:55:48	1	☐	
160	WRITE_MDB	U16	268566529	not available	user	WRITE_MDB	3	☐	change
161	READ_NUMBER	U16	268566530	not available	user	READ_NUMBER	1	☐	change

Рис. 35: *WEB-страница ПЛК*

Напишем еще одну программу, которая позволит управлять токовым выходом модуля расширения BRIC-AO-4 в соответствии с вводимым значением в единицах физической величины (mA). Для этого, сначала необходимо перевести вводимую физическую величину AO_out_x в единицы ЦАП из диапазона 0-4095, а затем, полученное значение AO_val_x необходимо передать в один из PDO-регистров AO_1_ao_val_x модуля расширения.

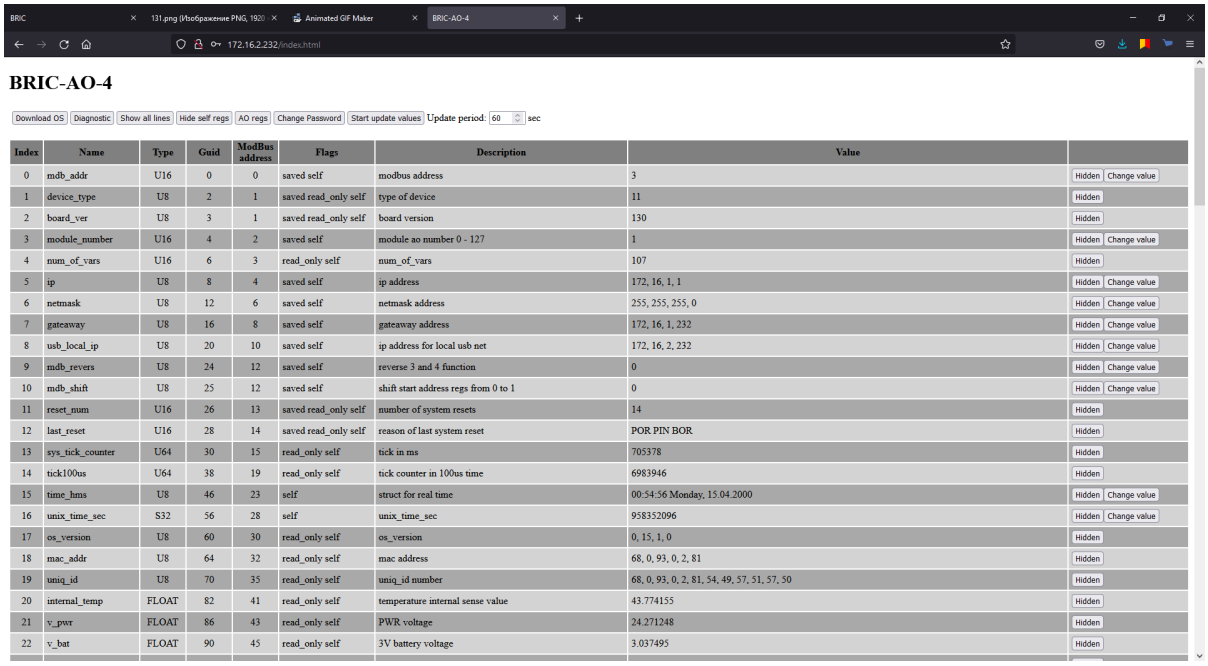
Расчет значения AO_val_x в единицах ЦАП осуществляется по формуле:

$$AO\_val\_x = (AO\_out\_x - AO\_calib\_b\_x) * AO\_calib\_a\_x;$$

где AO_out_x – вводимое значение тока (напряжения) выходного канала в mA (Вольтах);

AO_calib_a_x и AO_calib_b_x - индивидуальные калибровочные коэффициенты каждого канала.

В SDO-регистрах AO_1_ao_calib_a_x и AO_1_ao_calib_b_x модуля 1.1.x:AO_1 хранятся значения индивидуальных калибровочных коэффициентов AO_calib_a_x и AO_calib_b_x соответственно. Важно отметить, что для считывания значений калибровочных коэффициентов требуется установить статус read в поле Polling соответствующих SDO-регистров. Для первого выходного канала модуля 1.1.x:AO_1 программа будет иметь следующий вид:



BRIC-AO-4

Download OS | Diagnostic | Show all lines | Hide self regs | AO regs | Change Password | Start update values | Update period: 60 sec

Index	Name	Type	Guid	ModBus address	Flags	Description	Value	
0	mdb_addr	U16	0	0	saved self	modbus address	3	Hidden Change value
1	device_type	U8	2	1	saved read_only self	type of device	11	Hidden
2	board_ver	U8	3	1	saved read_only self	board version	130	Hidden
3	module_number	U16	4	2	saved self	module ao number 0 - 127	1	Hidden Change value
4	num_of_vars	U16	6	3	read_only self	num_of_vars	107	Hidden
5	ip	U8	8	4	saved self	ip address	172, 16, 1, 1	Hidden Change value
6	netmask	U8	12	6	saved self	netmask address	255, 255, 255, 0	Hidden Change value
7	gateaway	U8	16	8	saved self	gateaway address	172, 16, 1, 232	Hidden Change value
8	usb_local_ip	U8	20	10	saved self	ip address for local usb net	172, 16, 2, 232	Hidden Change value
9	mdb_revers	U8	24	12	saved self	reverse 3 and 4 function	0	Hidden Change value
10	mdb_shift	U8	25	12	saved self	shift start address regs from 0 to 1	0	Hidden Change value
11	reset_num	U16	26	13	saved read_only self	number of system resets	14	Hidden
12	last_reset	U16	28	14	saved read_only self	reason of last system reset	POR PIN BOR	Hidden
13	sys_tick_counter	U64	30	15	read_only self	tick in ms	705378	Hidden
14	tick100us	U64	38	19	read_only self	tick counter in 100us time	6983946	Hidden
15	time_hms	U8	46	23	self	struct for real time	00:54:56 Monday, 15.04.2000	Hidden Change value
16	unix_time_sec	S32	56	28	self	unix_time_sec	958352096	Hidden Change value
17	os_version	U8	60	30	read_only self	os_version	0, 15, 1, 0	Hidden
18	mac_addr	U8	64	32	read_only self	mac address	68, 0, 93, 0, 2, 81	Hidden
19	uniq_id	U8	70	35	read_only self	uniq_id number	68, 0, 93, 0, 2, 81, 54, 49, 57, 51, 57, 50	Hidden
20	internal_temp	FLOAT	82	41	read_only self	temperature internal sense value	43.774155	Hidden
21	v_pwr	FLOAT	86	43	read_only self	PWR voltage	24.271248	Hidden
22	v_bat	FLOAT	90	45	read_only self	3V battery voltage	3.037495	Hidden

Рис. 36: *WEB-страница модуля расширения*

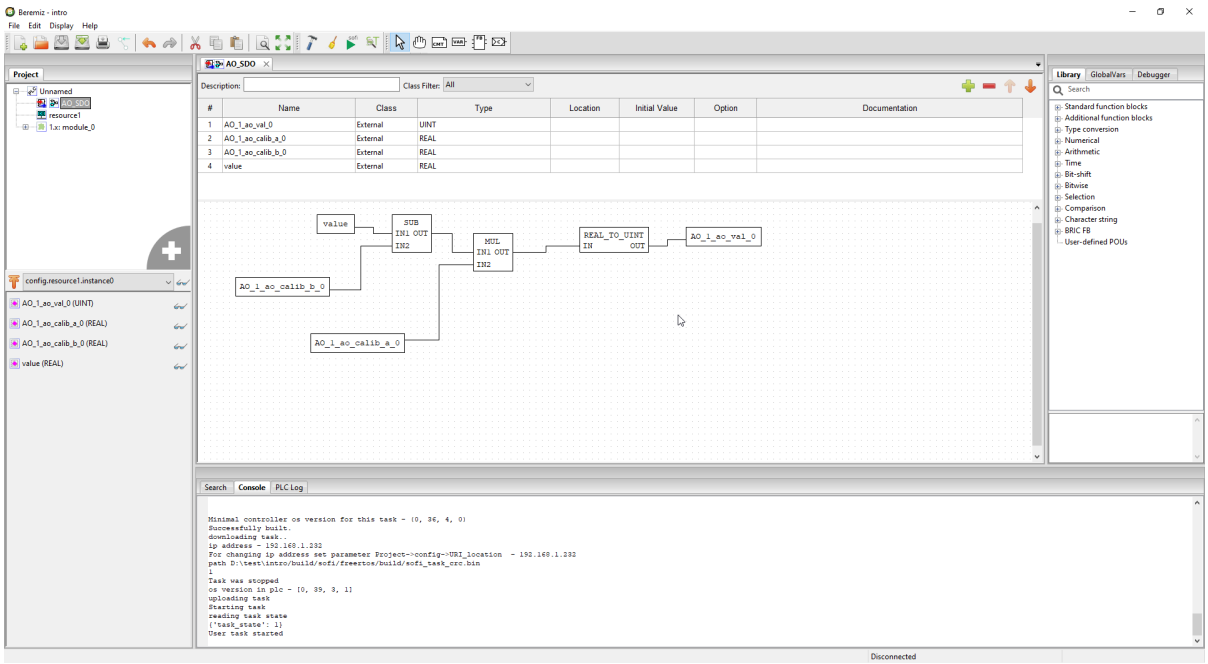


Рис. 37: *Программа на языке FBD*

После загрузки в ПЛК заходим в WEB-страницу и задаем какое-либо значение в mA в поле «value», например, «12».

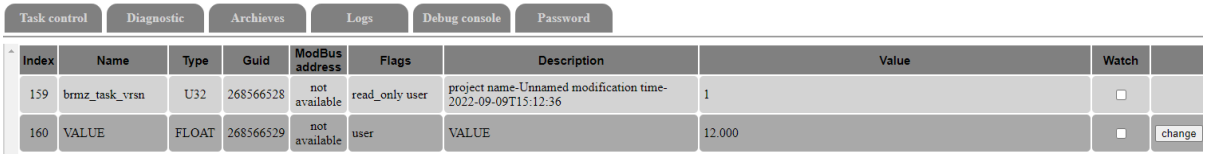


Рис. 38: *Параметр «value»*

На модуле расширения в нулевом канале будет задаваться аналоговый сигнал, равный 12mA. Данной программой мы можем перерасчитывать в значения ЦАП значения вводимых mA.

Примечание: Подробно о модуле расширения BRIC-AO-4 можно узнать по ссылке

3.7 Добавление нескольких модулей и назначение адресов (ручное/автоматическое)

Имеются некоторые особенности подключения модулей расширения и добавления их в Beremiz. Некоторые из них были упомянуты в разделе. На этом уроке добавим все модули расширения и научимся назначению адресов. Для начала подключим модули расширения к ПЛК. Порядок подключения не имеет значения.

Рис. 39: *Подключенные модули*

Модули в Beremiz необходимо добавить в порядке подключения.

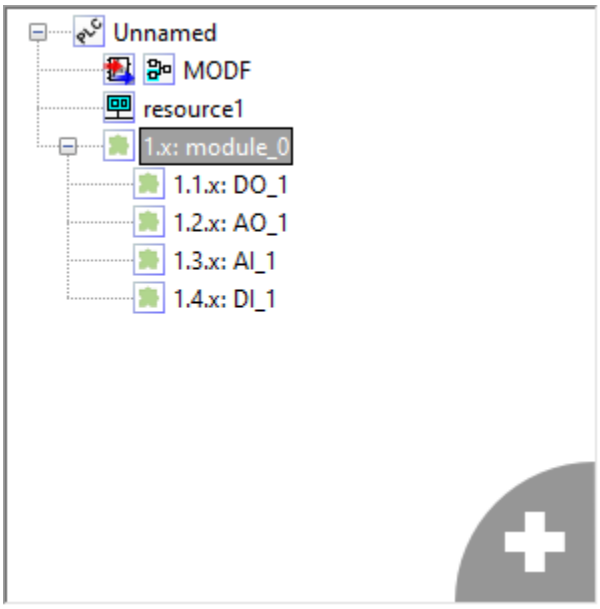


Рис. 40: *Добавленные модули*

Данные номера являются адресами CANOpen модулей, то есть для BRIC-DO-8 номером модуля является «1», для AO - «2» и так далее.

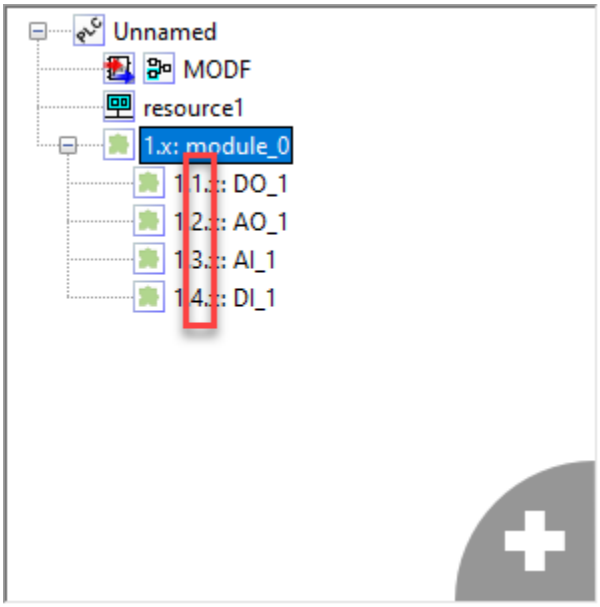


Рис. 41: *Номера подключенных модулей*

Данные адреса в самих модулях можно настроить двумя способами: вручную либо автоматически. Рассмотрим первый вариант.

Для начала заходим в WEB-страницу контроллера по IP-адресу http://192.168.1.232 и проверяем параметр *module_number*. В нашем случае он равен 127 и он не должен быть равен номерам модулей расширения.

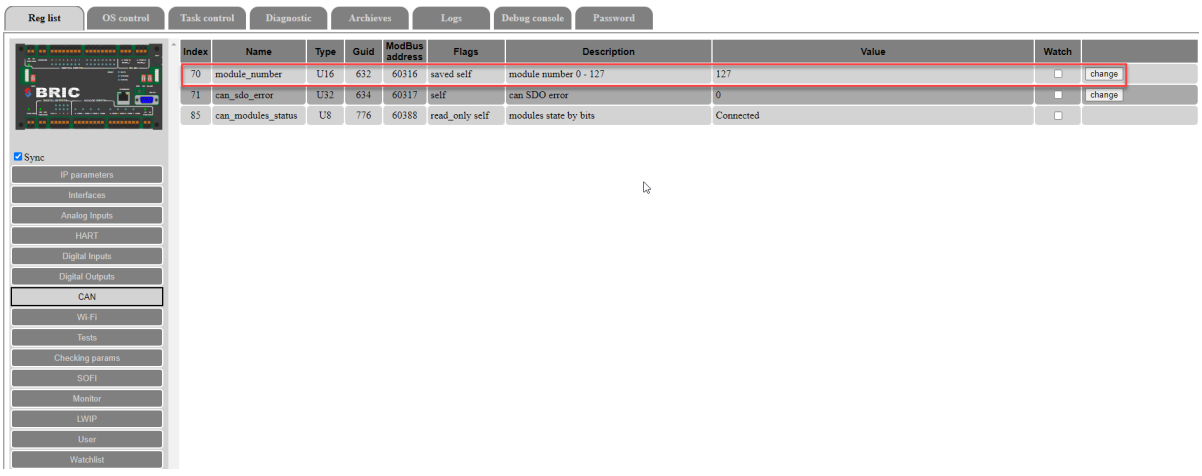
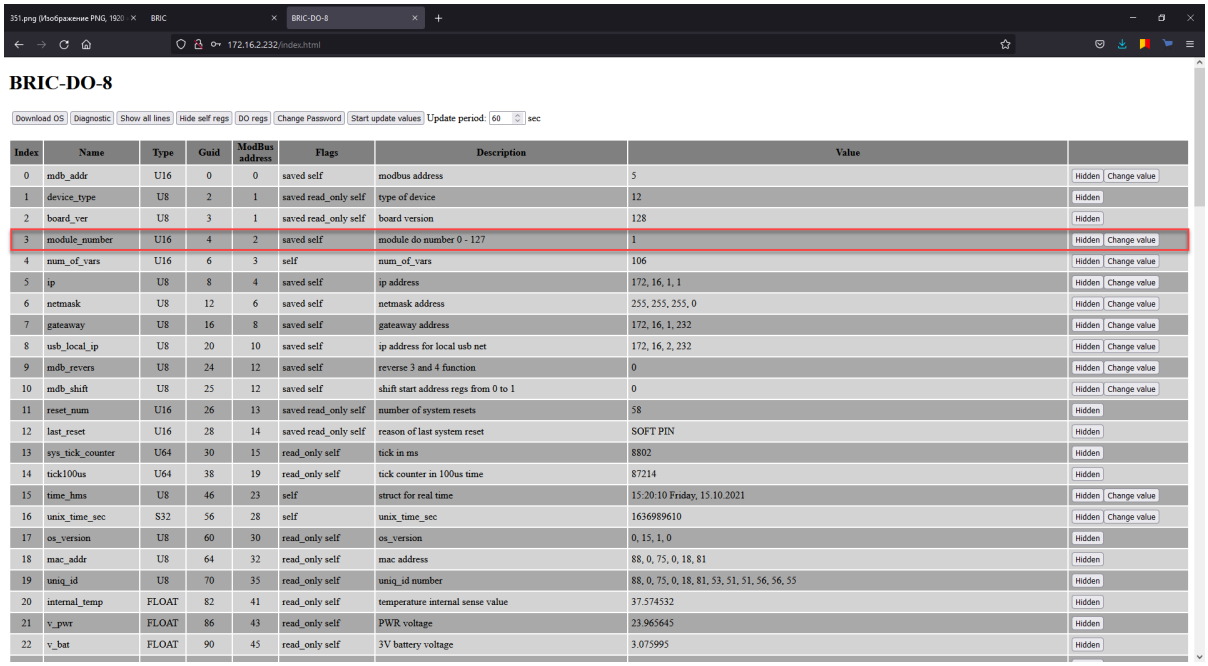


Рис. 42: *Module_number главного контроллера*

Далее подключаемся к модулю расширения BRIC-DO-8 через USB и заходим на его WEB-страницу по IP-адресу http://172.16.2.232. Нажимаем кнопку «Enter password» и вводим пароль «bric». Так как данный модуль является первым, вводим в поле «module_number» зачение «1», путём нажатия кнопки «Change value».

Аналогично последовательно настраиваем адреса остальных модулей расширения.



BRIC-DO-8

Download OS | Diagnostic | Show all lines | Hide self regs | DO regs | Change Password | Start update values | Update period: 60 sec

Index	Name	Type	Guid	ModBus address	Flags	Description	Value	
0	mdb_addr	U16	0	0	saved self	modbus address	5	Hidden Change value
1	device_type	U8	2	1	saved read_only self	type of device	12	Hidden
2	board_ver	U8	3	1	saved read_only self	board version	128	Hidden
3	module_number	U16	4	2	saved self	module do number 0 - 127	1	Hidden Change value
4	num_of_vars	U16	6	3	self	num_of_vars	106	Hidden Change value
5	ip	U8	8	4	saved self	ip address	172, 16, 1, 1	Hidden Change value
6	netmask	U8	12	6	saved self	netmask address	255, 255, 255, 0	Hidden Change value
7	gateaway	U8	16	8	saved self	gateaway address	172, 16, 1, 232	Hidden Change value
8	usb_local_ip	U8	20	10	saved self	ip address for local usb net	172, 16, 2, 232	Hidden Change value
9	mdb_revers	U8	24	12	saved self	reverse 3 and 4 function	0	Hidden Change value
10	mdb_shift	U8	25	12	saved self	shift start address regs from 0 to 1	0	Hidden Change value
11	reset_num	U16	26	13	saved read_only self	number of system resets	58	Hidden
12	last_reset	U16	28	14	saved read_only self	reason of last system reset	SOFT PIN	Hidden
13	sys_tick_counter	U64	30	15	read_only self	tick in ms	8802	Hidden
14	tick100us	U64	38	19	read_only self	tick counter in 100us time	87214	Hidden
15	time_hms	U8	46	23	self	struct for real time	15:20:10 Friday, 15.10.2021	Hidden Change value
16	unix_time_sec	S32	56	28	self	unix_time_sec	1636989610	Hidden Change value
17	os_version	U8	60	30	read_only self	os_version	0, 15, 1, 0	Hidden
18	mac_addr	U8	64	32	read_only self	mac address	88, 0, 75, 0, 18, 81	Hidden
19	uniq_id	U8	70	35	read_only self	uniq_id number	88, 0, 75, 0, 18, 81, 53, 51, 51, 56, 56, 55	Hidden
20	internal_temp	FLOAT	82	41	read_only self	temperature internal sense value	37.574532	Hidden
21	v_pwr	FLOAT	86	43	read_only self	PWR voltage	23.965645	Hidden
22	v_bat	FLOAT	90	45	read_only self	3V battery voltage	3.075995	Hidden

Рис. 43: *Module_ number модуля расширения BRIC-DO-8*

Теперь рассмотрим автоматическое назначение адресов модулей.

После подключения модулей настроим их с помощью джампера SW-2. Сначала для BRIC-DO-8 переводим состояние переключателей в SW-1 > OFF, SW-2 > ON и нажимаем кнопку RESET. Единовременно на межмодульной CAN-шине может быть только одно устройство в режиме получения нового адреса. После успешного получения нового адреса светодиод CONNECT загорится оранжевым цветом, что будет свидетельствовать о наличии обмена по CAN-интерфейсу. После успешного присвоения нового адреса необходимо вернуть модуль в нормальный режим работы SW-1 > OFF, SW-2 > OFF.

Аналогично последовательно настраиваем адреса остальных модулей расширения.

3.8 Тестовая программа. ПИД-регулятор

Итак, мы научились подключать модули, добавлять их в Beremiz и настраивать адреса. Для закрепления пройденного, а также для ознакомления с некоторыми блоками в данном уроке мы рассмотрим ПИД-регулятор.

Пропорционально-интегрально-дифференцирующий (ПИД) регулятор — устройство в управляющем контуре с обратной связью. Используется в системах автоматического управления для формирования управляющего сигнала с целью получения необходимых точности и качества переходного процесса. ПИД-регулятор формирует управляющий сигнал, являющийся суммой трёх слагаемых, первое из которых пропорционально разности входного сигнала и сигнала обратной связи (сигнал рассогласования), второе — интегралу сигнала рассогласования, третье — производной сигнала рассогласования.

Подключим к ПЛК модули расширения BRIC-AO-4 и BRIC-AI-16. Добавляем их в Beremiz. Язык программирования выберем FBD для наглядности. Функциональный блок PID выбирается из «Additional function blocks».

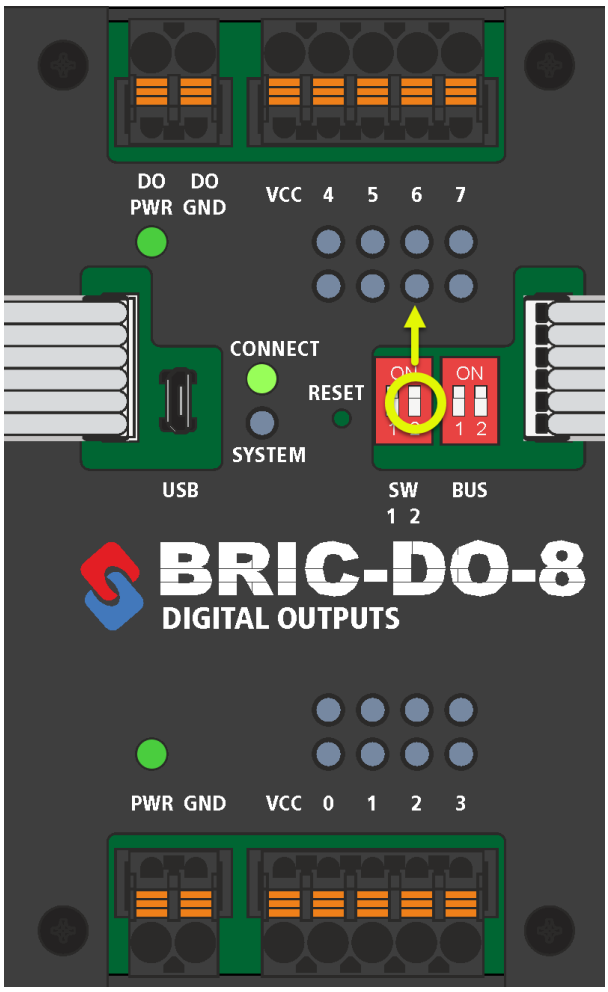


Рис. 44: *Переключатель SW-2*

#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	PID0	Local	PID				

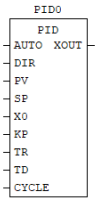


Рис. 45: *Функциональный блок PID*

Вход	Тип	Описание
AUTO	BOOL	Включение режима ПИД-регулятора (0 - ручное регулирование, 1 - автоматическое)
DIR	BOOL	Направление регулирования (0 - обратное регулирование, 1 - прямое. По умолчанию стоит прямое регулирование)
PV	REAL	Входной сигнал (автоматический режим)
SP	REAL	Заданное значение (уставка в автоматическом режиме)
X0	REAL	Заданное значение (уставка в ручном режиме)
KP	REAL	Пропорциональный коэффициент
TR	REAL	Интегральный коэффициент
TD	REAL	Дифференциальный коэффициент
CYCLE	TIME	Время цикла
Выход	**Тип**	**Описание**
XOUT	REAL	Выход ПИД-регулятора

Добавим переменные *AO_1_ao_val_0* и *AI_1_ai_physical_0* в программу. В нашем случае задание PV будем получать от AI_1_ai_physical а выход ПИД-регулятора XOUT будет подключен к AO_1_ao_val. Добавим также остальные входные переменные. Также для нашей программы нужны будут SDO-регистры модуля AO, а именно калибровочные коэффициенты нулевого канала AO_1_ao_calib_a_0 и AO_1_ao_calib_b_0. Копируем регистры в программу, в «Polling» пишем «read».

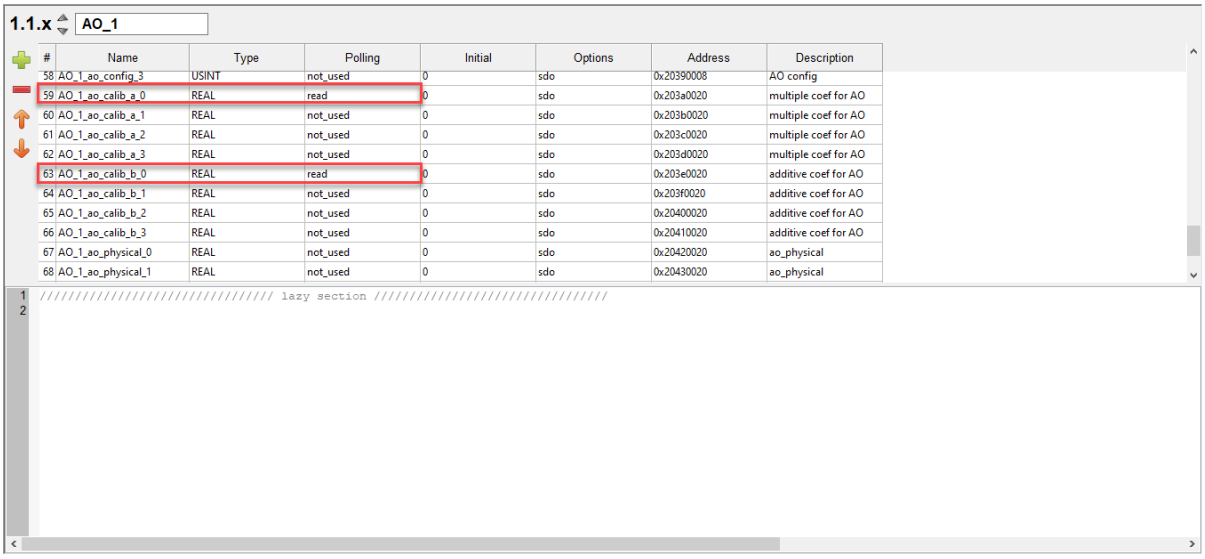


Рис. 46: *SDO-регистры модуля AO*

Примечание: Расчет необходимого значения AO_val_x осуществляется по формуле:

AO_val_x = (AO_OUT - AO_calib_b_x) * AO_calib_a_x

где AO_OUT - необходимое выходное значение в «мА» или «В», AO_calib_a_x, AO_calib_b_x - индивидуальные калибровочные коэффициенты каждого канала.

Параметр	Значение по умолчанию	Диапазон	Описание
AO_val_x	-	0 – 4095	Выходное значение аналогового канала в единицах АЦП
AO_calib_a_x	163.8 - для токовых каналов / 287.368 - для каналов напряжения	-	Калибровочный коэффициент A
AO_calib_b_x	0.0 - для токовых каналов / 0.0 - для каналов напряжения	-	Калибровочный коэффициент B

Для того чтобы ручное значение записывалось в mA необходимо вычислить его с помощью формулы:

X0 = (manual_set - AO_calib_b_x) * AO_calib_a_x

Реализация на языке FBD представлена на рисунке ниже

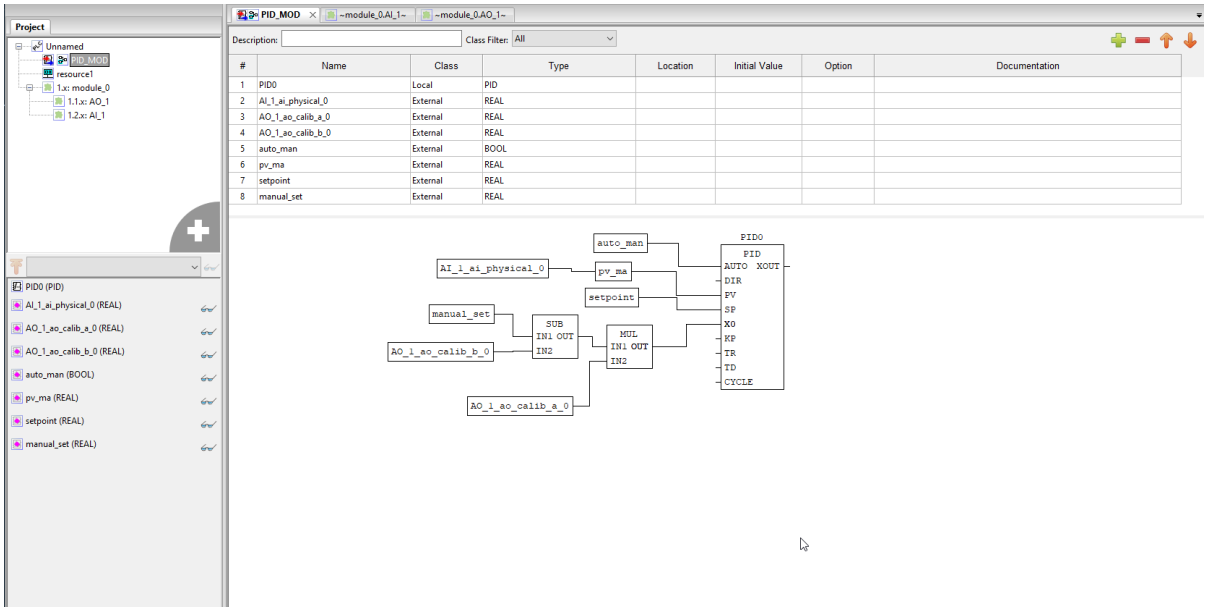


Рис. 47: *Подключение manual_set*

Добавим Пропорциональный, Интегральный и Дифференциальные коэффициенты. Класс выбираем так же как и у других «External», для взаимодействия через WEB-страницу.

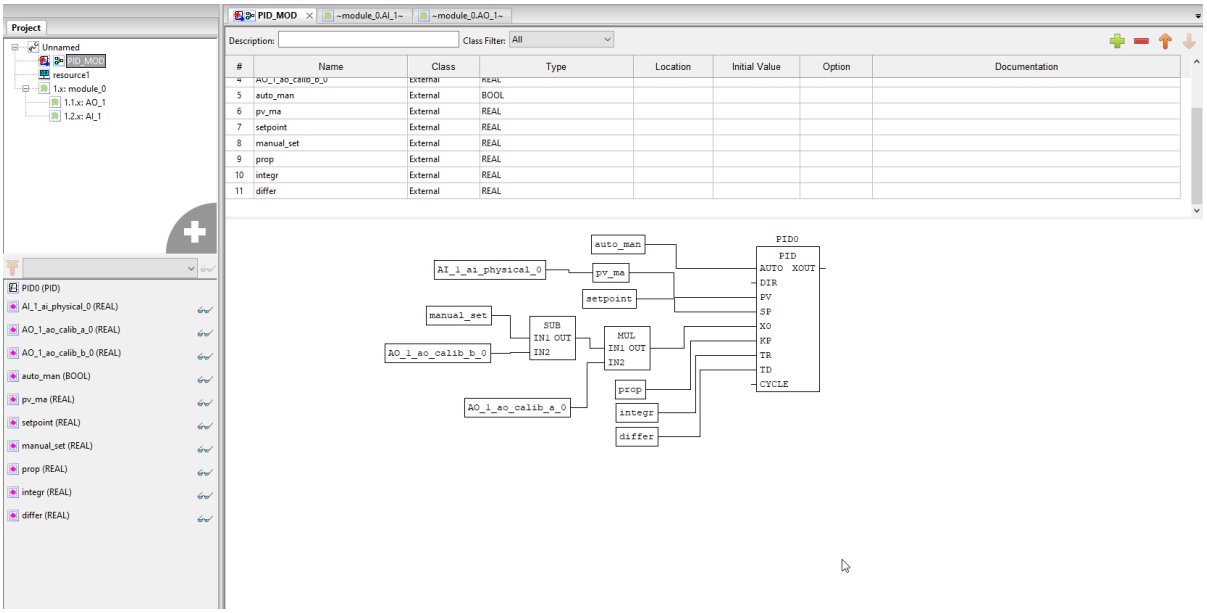


Рис. 48: *Подключение коэффициентов*

Для ПИД-регулятора выберем время цикла равную в 1 секунду. Прописываем «T#1s» в колонке «Initial Value».

Внимание: Тип данных «Time» не является внешним, поэтому класс необходимо выбрать «Local»

Осталось подключить выход ПИД-регулятора. XOUT преобразуется в тип данных «UINT» и записывается в AO_1_ao_val_0. Чтобы выход выдавался в mA, добавим коэффициенты. В данном случае используется обратная формула:

$$xout\_ma = (XOUT / AO\_calib\_a\_x) + AO\_calib\_b\_x;$$

Добавим переменные в главный проект. Опишем переменные в разделе «Documentation». Программа готова.

Подключим нулевой канал модулей расширения между собой. Финальный результат на рисунке ниже.

Загружаем прошивку и заходим в WEB-страницу. Задаем *manual_set* значение в 5 mA. В данном случае ПИД-регулятор находится в ручном режиме и выход равен заданию.

Задаем параметры автоматического режима. После задания уставок и коэффициентов, задаем «auto_man» значение «1» тем самым запускаем ПИД-регулятор в автоматическом режиме. Выход регулятора XOUT стремится к уставке.

После достижения заданной уставки, ПИД-регулятор удерживает выходное значение в пределах данного задания.

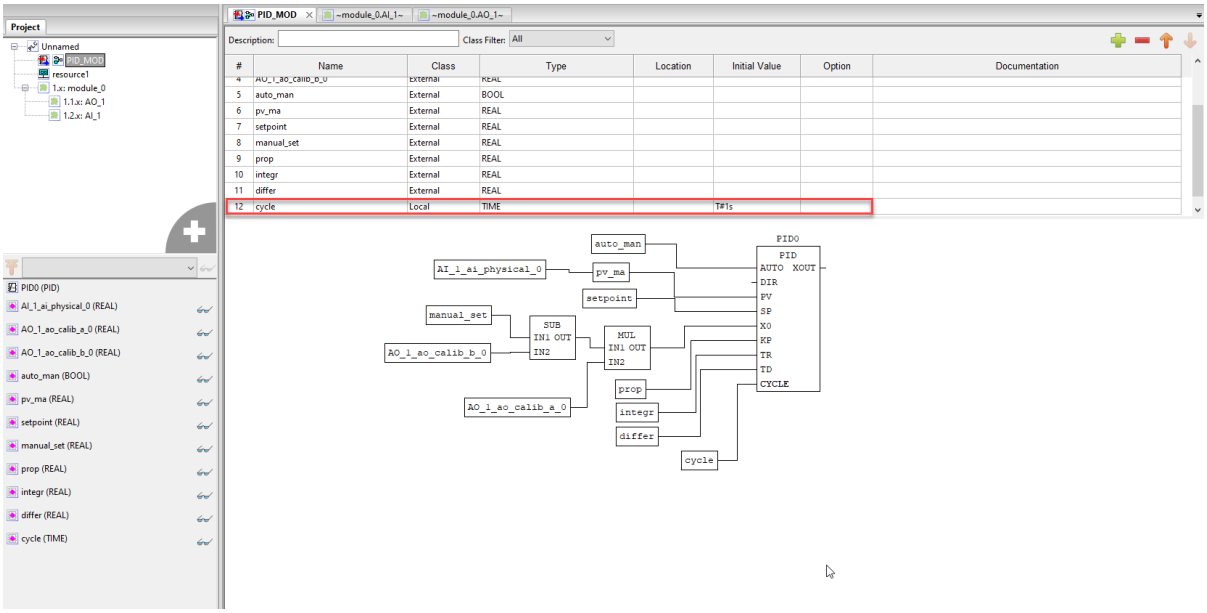


Рис. 49: *Добавление времени цикла*

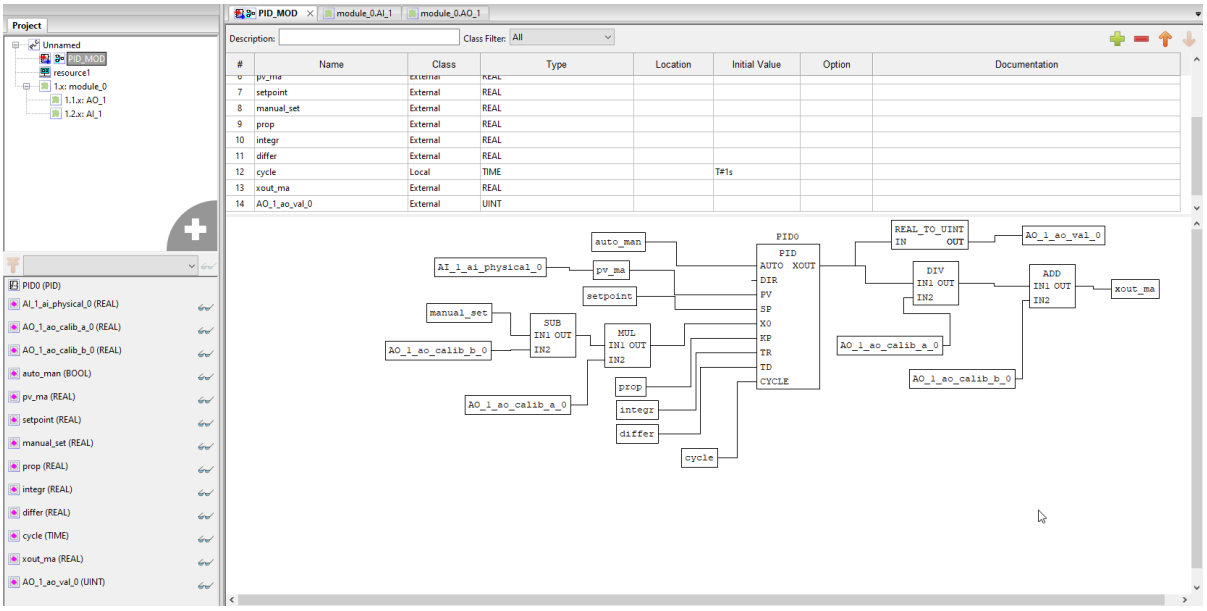


Рис. 50: *Подключение выхода XOUT*

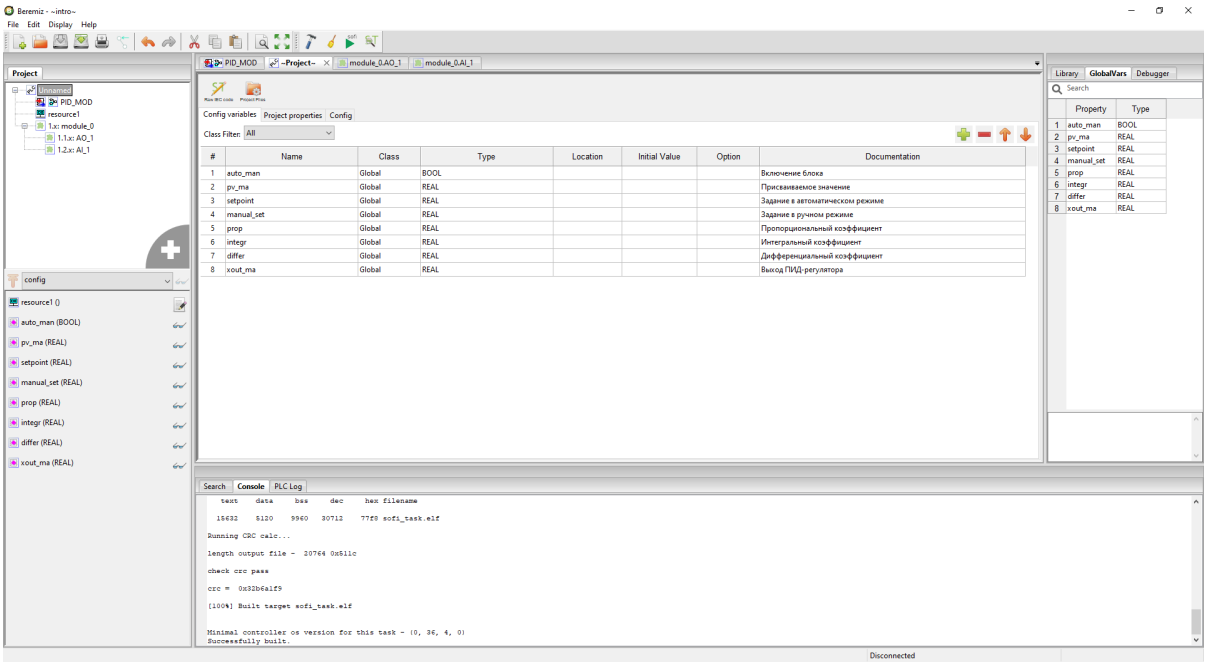


Рис. 51: *Программа ПИД-регулятора*

Рис. 52: *Подключение модулей расширения*

Рис. 53: *WEB-страница контроллера*

Рис. 54: *ПИД-регулятор в ручном режиме*

Рис. 55: *ПИД-регулятор в автоматическом режиме*

Рис. 56: *ПИД-регулятор в автоматическом режиме после достижения уставки*

Рис. 57: *Результат программы*

Глава 4

Modbus

4.1 Основная информация. Добавление модуля Modbus

Modbus — открытый коммуникационный протокол, основанный на архитектуре «клиент-сервер». Широко применяется в промышленности для организации связи между электронными устройствами. Может использоваться для передачи данных через последовательные линии связи RS-485, RS-422, RS-232, а также сети TCP/IP (Modbus TCP).

ПЛК BRIC обладает возможностью опроса данных по протоколу Modbus, ретранслировать данные из одного канала в другой, а также имеется возможность расширения адресного пространства. На борту у контроллера имеется 2 порта RS-482 и 1 порт RS-232.

По умолчанию, Modbus адресное пространство имеет зарезервированную область адресов, которая показана на рисунке ниже.

Для обмена со сторонними устройствами по Modbus необходимо добавить подмодуль «ModbusRTUMaster». Для подключения необходимо добавить его в элемент «Modbus support», щелкнув правой клавишей мыши и выбрав пункт «Add ModbusRTUMaster». Затем в окне конфигурации настроить канал опроса. Для каждого физического канала может быть не более одного элемента «ModbusRTUMaster».

Для увеличения адресного пространства каждого типа регистров (Coils, Input Discrete, Input Registers, Holding Registers) необходимо добавить подмодуль «MemoryArea». Для подключения «MemoryArea» необходимо подвести курсор к созданной ветке «Modbus support», щелкнуть правой клавишей мыши и выбрать пункт «Add MemoryArea».

Для ретранслирования пакетов из одного канала в другой, а также для приёма-передачи пакетов Modbus TCP в Modbus RTU и Modbus RTU – Modbus RTU необходио подключить подмодуль «ModbusRoute». Для его подключения необходимо подвести курсор к созданной ветке «Modbus support», щелкнуть правой клавишей мыши и выбрать пункт «Add ModbusRoute».

Более подробную информацию о подключении подмодулей мы пройдем в последующих уроках.

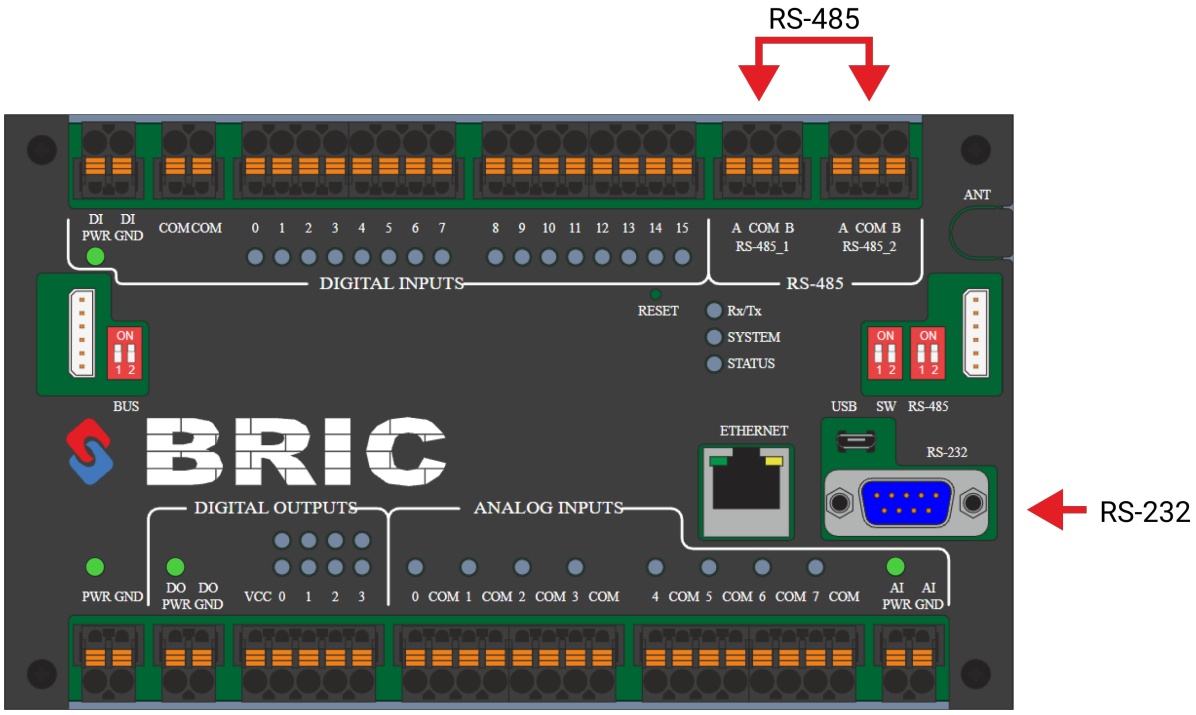


Рис. 1: Порты Modbus ПЛК BRIC

Адресное пространство Modbus	
0x30000–0x39999	Область адресов, выделенных под чтение и запись массивов пользовательской программы
0x40000–0x49999	Область адресов, выделенных под чтение и запись регистров пользовательской программы
0x60000–0x69999	Область адресов, выделенных под системные регистры ПЛК

Рис. 2: Добавление подмодуля ModbusRTUMaster

Рис. 3: Добавление подмодуля MemoryArea

Рис. 4: Добавление подмодуля ModbusRoute

4.2 Структура записи адреса Modbus. Присваивание глобальным переменным Modbus-адреса

Для обмена контроллера данными с подчиненными modbus-устройствами, необходимо создать запрос определенного типа к нужным регистрам памяти modbus-устройств и объявить в проекте глобальные переменные, связанные с соответствующими адресами этих регистров. В данном уроке разберем все нюансы присваивания для разных подмодулей Modbus.

Адрес глобальной переменной присваивается внесением записи в поле «Location». Структура записи выглядит следующим образом:

%[Форма] [Размер] [Идентификатор] . [Номер]

Таблица 1: **Форма регистра**

Тип формы	Описание
Q	Глобальная переменная используется для записи Reg/Coil slave–устройства (WriteSingleCoil, WriteSingleRegister, WriteMultipleCoils, WriteMultipleRegisters)
I	Глобальная переменная используется для чтения Reg/Coil slave–устройства (ReadCoils, ReadInputDiscretes, ReadHoldingRegisters, ReadInputRegisters)
M	Глобальная переменная используется для записи и чтения

Таблица 2: **Размер переменной**

Размер	Количество байтов	Тип данных
D	4	DINT, REAL, UDINT, DWORD
L	8	LINT, ULINT, LREAL, LWORD
B	1	BYTE, USINT, SINT
X	1	BOOL
W	2	WORD, INT, UINT

Таблица 3: **Идентификатор элемента**

Структура	Предназначение
X.X.X	Для ModbusRequest
X.X	Для MemoryArea
X	Для остальных, не входящих в модуль расширения

Номер регистра выставляется согласно номеру в выборке ModbusRequest, при этом номер первого регистра равен 0.

Если пока ничего не понятно, то не стоит переживать, сейчас мы на примерах узнаем подробнее.

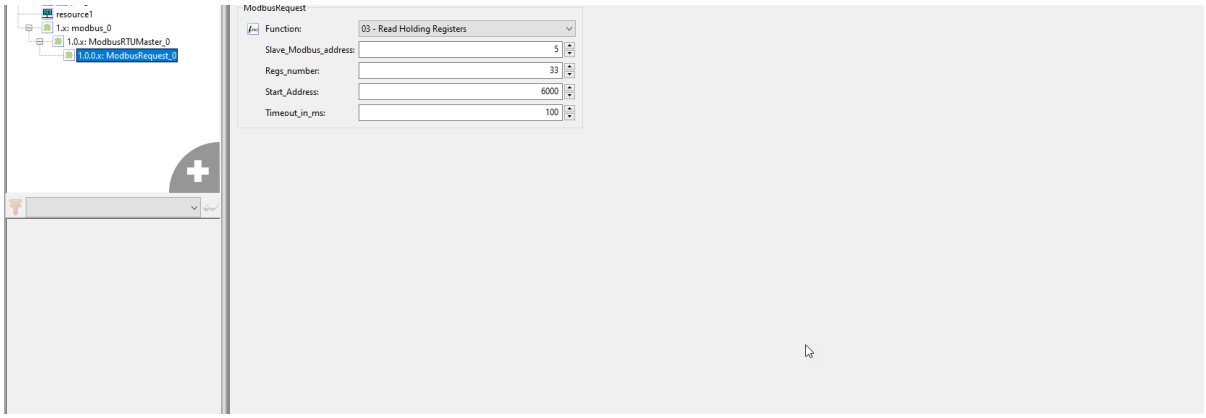
Допустим, есть slave–устройство, подключенное к порту RS-485_1 и нам необходимо прочитать состояние регистра с типом данных UINT по адресу 6032 и 6015. Modbus-адрес устройства - 5. Скорость передачи данных - 9600, 8-n-1.

Добавим подмодуль ModbusRTUMaster.

Примечание: Более подробно о подмодуле ModbusRTUMaster можно узнать в *Подмодуль ModbusRTUMaster. Добавление и конфигурирование ModbusRequest.*

Выбираем нужный нам порт - RS_485_1, скорость передачи данных и тип 8-n-1. Далее добавляем ModbusRequest для настройки конкретного slave–устройства.

Создадим запрос типа «Read Holding Registers» к нашему modbus-устройству, имеющему modbus-адрес 5 (Slave_Modbus_Adress:5). Начальный адрес пула регистров Start_Address сделаем равным: 6000. Так как регистр с адресом 6032 имеет размер W (используется 2 байта для хранения данных типа UINT, т.е. одна ячейка), то Regs_number запишем как 33. Таким образом, задействуются регистры с 6000 по 6032 включительно.



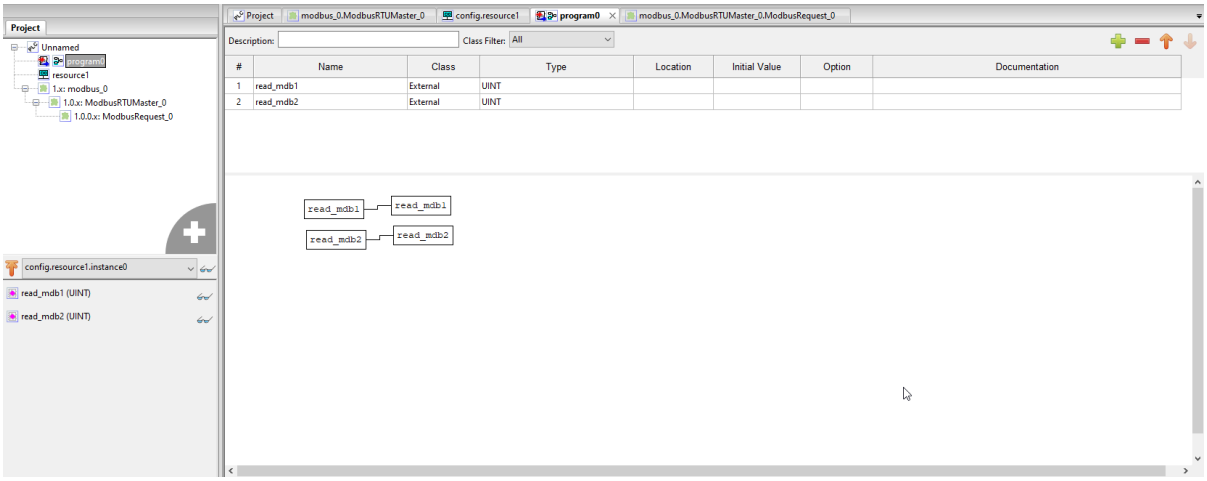
Далее напишем программу на языке FBD. Добавим глобальные переменные *read_mdb1* и *read_mdb2* с типом данных UINT.

Далее переходим во вкладку с глобальными переменными проекта. Пропишем Modbus-адреса для наших переменных.

Как видно, мы задали адрес по структуре, указанной выше:

- Прописываем «%»
- Прописываем форму регистра для чтения ReadHoldingRegisters - «I»
- Прописываем размер переменной - «W» в соответствии с типом данных UINT
- Для ModbusRequest структура имеет вид «X.X.X» - «1.0.0.X»
- Прописываем номера необходимых регистров 32 и 15.

Совет: Если в проекте несколько ModbusRequest, необходимо прописать нужный субмодуль, проверяя



в дереве проекта

Имеется еще один вариант прописывания адреса. Для этого нажимаем на три точки в «Location» и далее выбираем наши нужные регистры.

В данном случае мы сами выбираем адрес из нужного подмодуля. Также не забываем указать правильный номер регистра.

Внимание: После ввода Modbus-адреса через три точки в «Location» обычно ИСР Beremiz по умолчанию меняет тип данных на WORD

Рассмотрим другой пример. Допустим, у нас есть несколько переменных, которые необходимо прочитать другому устройству. В данном случае наш контроллер будет выполнять функции slave–устройства.

Итак, пусть переменными у нас будут *LocalVar0 - LocalVar3* с типами данных *WORD*, *REAL*, *LWORD* и *BOOL*. Напишем простую программу, которая будет фиктивной.

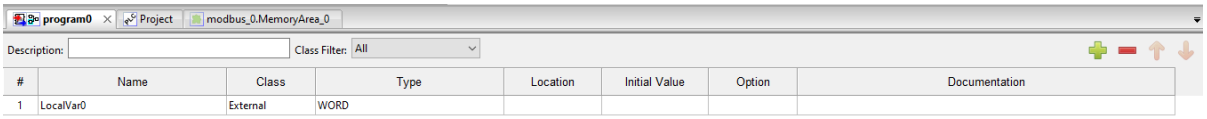
Остальные переменные добавим в *Config variables*.

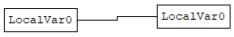
Следующим шагом мы добавим подмодуль MemoryArea. Зададим область адресов с 6000 с количеством регистров - 10. Тип MemoryArea выберем 03 - HoldingRegisters.

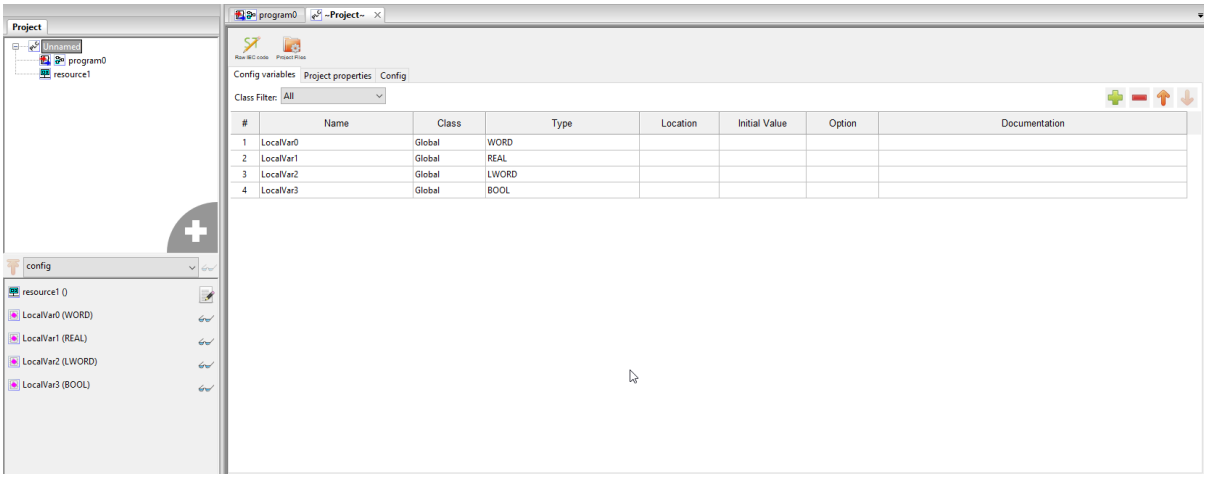
Примечание: Более подробно о подмодуле ModbusRTUMaster можно узнать в *Добавление и конфигурирование подмодуля MemoryArea*

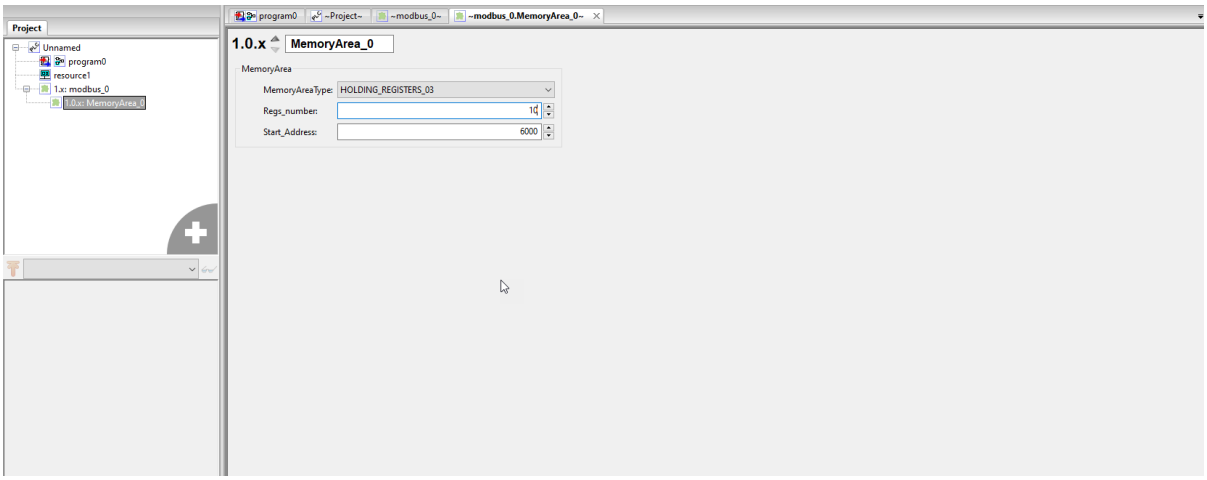
Остался последний шаг - корректно прописать адреса. С помощью таблицы структуры мы запишем сначала для *LocalVar0*:

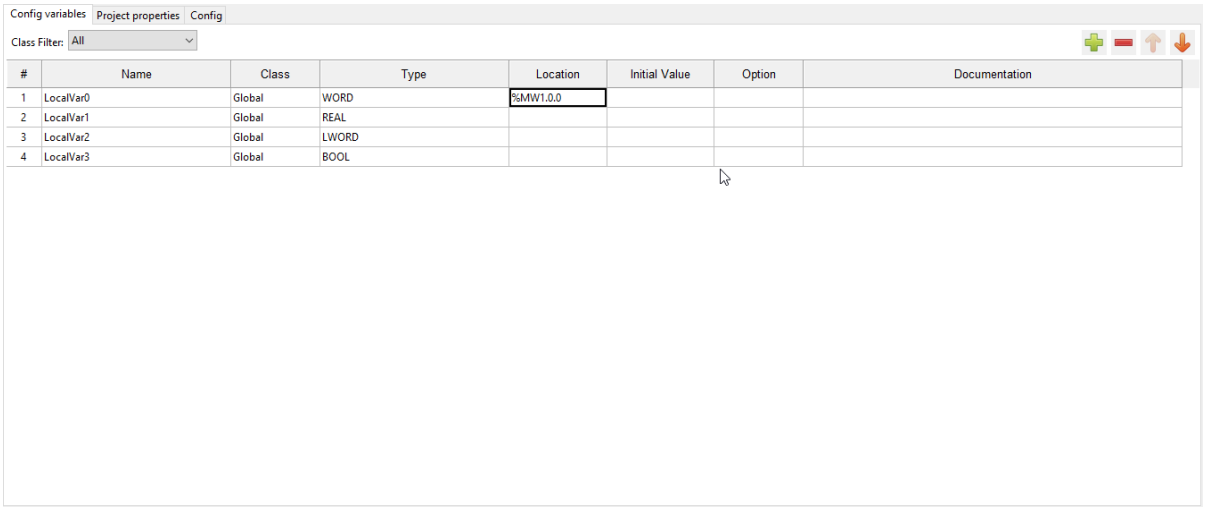
- Прописываем «%»
- Прописываем форму регистра - «M»
- Прописываем размер переменной - «W» в соответствии с типом данных WORD
- Для MemoryArea структура имеет вид X.X - «1.0»
- Прописываем номер, пусть *LocalVar0* будет первым - «1.0.0»


program0
Project
modbus_0.MemoryArea_0
Description:
Class Filter:
All
Name Class Type Location Initial Value Option Documentation
1 LocalVar0 External WORD

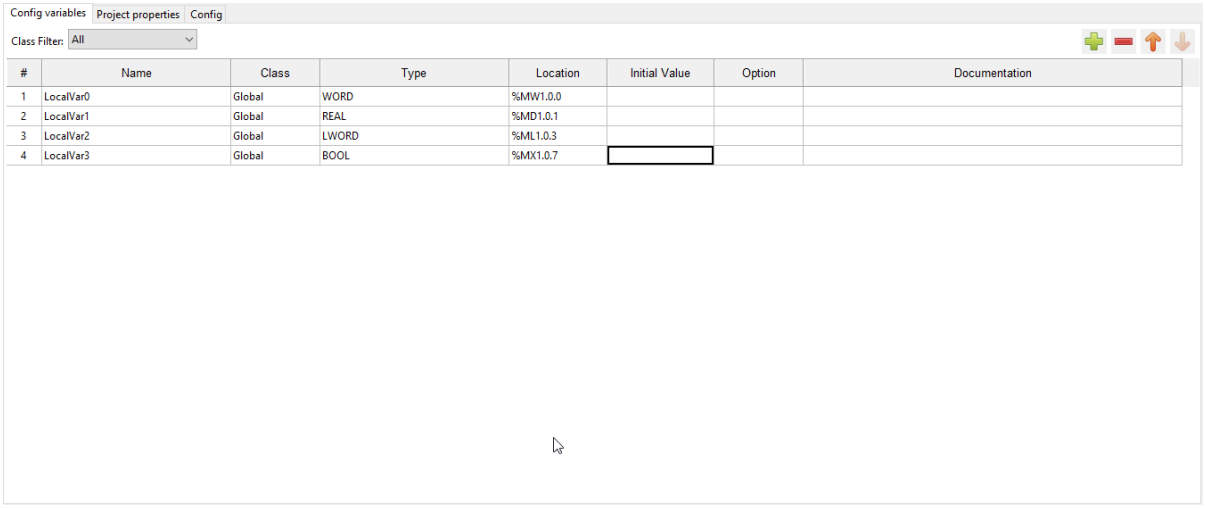

LocalVar0
LocalVar0


Project
Unnamed
program0
resource1
config
resource1 ()
LocalVar0 (WORD)
LocalVar1 (REAL)
LocalVar2 (LWORD)
LocalVar3 (BOOL)
program0
~Project~
Config variables
Project properties
Config
Class Filter:
All
Name Class Type Location Initial Value Option Documentation
1 LocalVar0 Global WORD
2 LocalVar1 Global REAL
3 LocalVar2 Global LWORD
4 LocalVar3 Global BOOL


Project
Unnamed
program0
resource1
1.x: modbus_0
1.0.x: MemoryArea_0
program0
~Project~
~modbus_0~
~modbus_0.MemoryArea_0~
1.0.x
MemoryArea_0
MemoryArea
MemoryAreaType:
HOLDING_REGISTERS_03
Regs_number:
1
Start_Address:
6000



Итоговый вариант глобальных пременных выглядит так:



Возникает вопрос: почему *LocalVar2* прописан с адресом %ML1.0.3, а не %ML1.0.2? Потому что тип данных REAL имеет количество битов 4, значит занимает 2 ячейки. Аналогично с этим и прописан адрес для *LocalVar3* - количество битов для типа данных LWORD - 8.

Готово, компиллируем и загружем программу, и теперь мастер-устройство может опросить, менять данные переменные обратишись к конроллеру по Modbus-адресу - 3(по умолчанию) со скоростью - 115200 к регистрам с адресами от 6000.

4.3 Подмодуль ModbusRTUMaster. Добавление и конфигурирование ModbusRequest.

В данном уроке рассмотрим подключение ModbusRTUMaster. Как говорилось ранее, ПЛК BRIC имеет возможность обращаться к slave-устройствам в качестве мастера. Для реализации нам потребуется программа Modbus Slave. Добавим подмодуль ModbusRTUMaster. Рассмотрим подробнее элементы подмодуля.

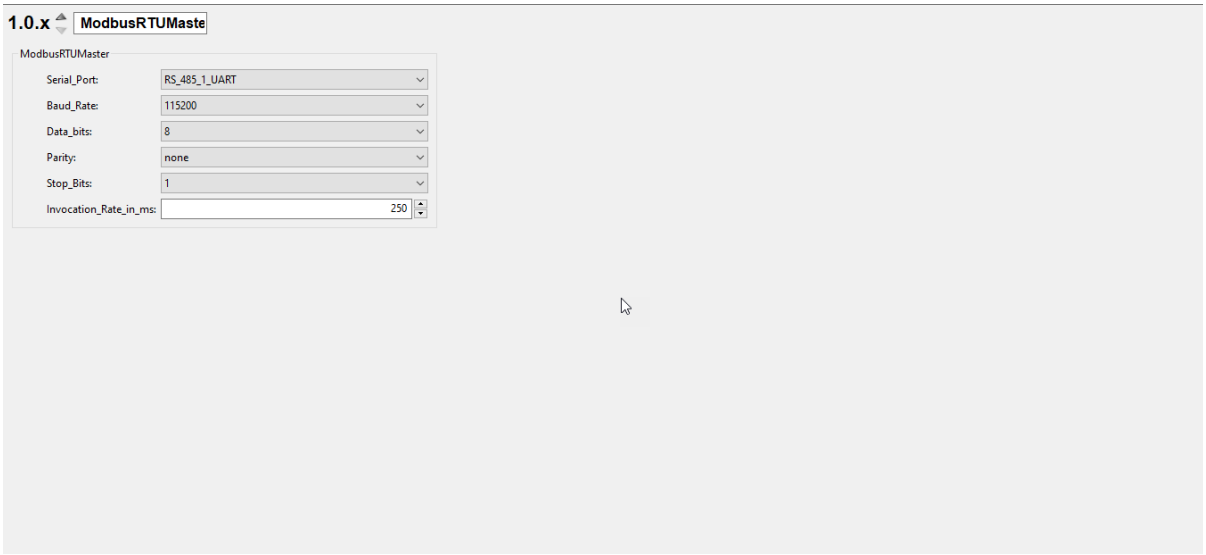
Таблица 4: Элементы ModbusRTUMaster

Элемент	Описание
Serial_Port	**Выбор последовательного порта:** • RS_232_UART • RS_485_1_UART • RS_485_2_UART • RS_485_IMMO_UART
Baud_Rate	**Выбор скорости передачи:** • 1200 • 2400 • 4800 • 9600 • 14400 • 19200 • 28800 • 38400 • 56000 • 57600 • 76800 • 115200 • DEFAULT_BAUD_RATE
Data_bits	**Выбор количества байт данных:** • 7 • 8 • 9
Parity	**Выбор паритета:** • even (четный) • odd (нечетный) • none (без проверки)
Stop_Bits	**Выбор количества стоп-битов:** • 1 • 2
Invocation_Rate_in_ms	Скорость вызова, мс

См.также:

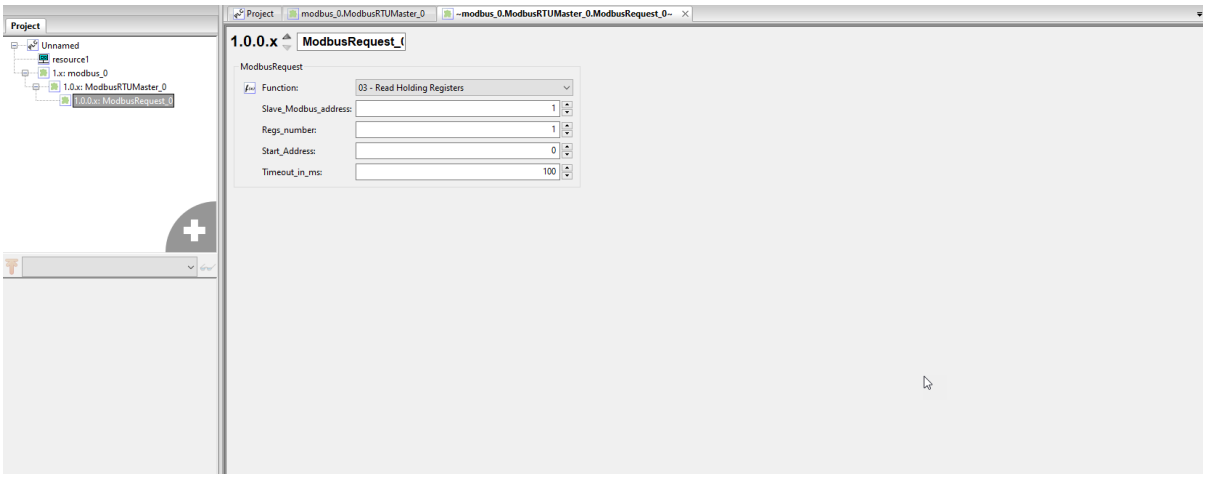
Подробнее можно узнать по ссылке

В нашем уроке мы выберем настройки, представленные на рисунке ниже.



То есть, порт мы выбираем RS_485_1, скорость передачи данных - 115200, количество битов данных - 8, без проверки на четность/нечетность, с количеством стоповых битов - 1.

Далее мы добавим субмодуль ModbusRequest.



Рассмотрим подробнее элементы ModbusRequest.

Таблица 5: Элементы ModbusRequest

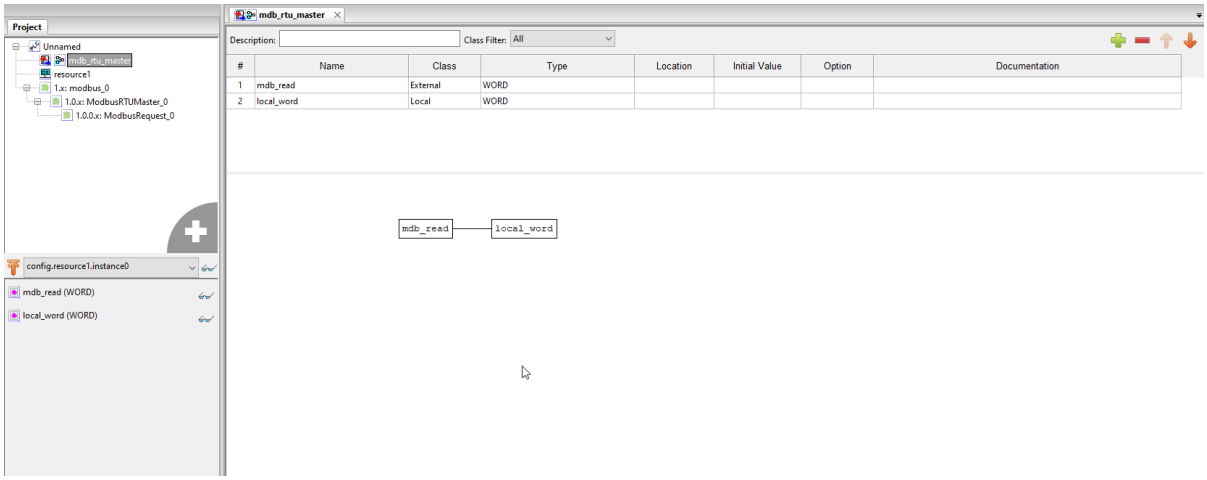
Элемент	Описание
Function	**Выбор функции:** • ReadCoils — Функция используется для получения состояний определенного количества реле, начиная с указанного в запросе. Состояние одного реле при этом передается одним битом. Если бит установлен в 1 – реле включено, если 0 – реле отключено. • ReadInputDiscretes — Функция используется для получения состояний определенного количества дискретных входов, начиная с указанного в запросе. Состояние одного входа при этом передается одним битом. Если бит установлен в 1 – вход замкнут, если 0 – вход разомкнут. • ReadHoldingRegisters — Функция используется для чтения указанного количества 2–Байтных регистров. • ReadInputRegisters — Функция используется для получения состояний определенного количества 2–Байтных регистров, хранящих состояние дискретных входов, начиная с указанного в запросе. Значение одного регистра передается двумя байтами. • WriteSingleCoil — Функция используется для включения/отключения одного реле. Требуемое состояние реле передается двумя байтами. • WriteSingleRegister — Функция выполняет запись нового значения в указанный регистр. • WriteMultipleCoils — Функция используется для групповой установки состояний определенного количества реле, начиная с указанного. Состояние одного реле при этом передается одним битом. Если бит установлен в 0 – реле отключено, если 1 – реле включено. • WriteMultipleRegisters — Функция выполняет запись новых значений в указанные регистры
Slave_Modbus_address	Выбор адреса slave-устройства (0-255)
Regs_number	Выбор количества регистров (1-1600)
Start_Address	Выбор начального адреса (0 - 65535)
Timeout_in_ms	Выбор времени, необходимое для ответа slave–устройству, мс

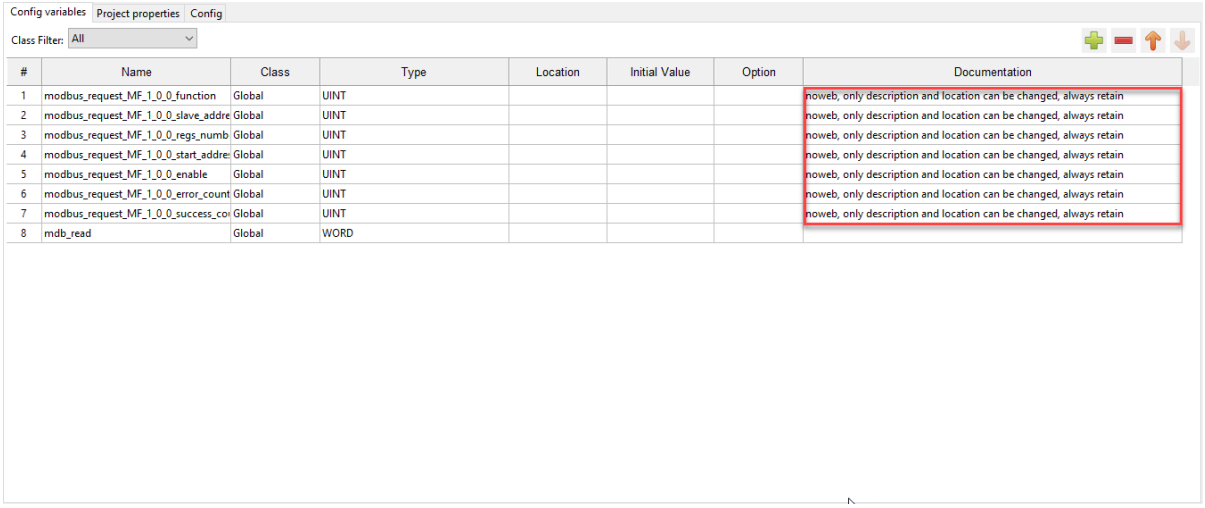
Для нашего урока мы определим настройки, представленные ниже.

Будет произведено чтение 2-байтных регистров начиная с адреса 5000 slave–устройства по Modbus-адресу 10. Напишем программу на языке FBD. Определим глобальную переменную *mdb_read* формата *WORD* и локальную переменную *local_word*. Данная программа будет фиктивной, так как основную функцию на себя возьмет ModbusRTUMaster.

Заходим в глобальные настройки проекта. Можно заметить, что после добавления подмодуля ModbusRTUMaster автоматически добавились его регистры. В разделе «Documentation» можно увидеть параметры регистров.

Примечание: Если в разделе «Documentation» прописать «noweb» - данный регистр не будет отоб-


Project
Unnamed
mdb_rtu_master
resource1
1.x: modbus_0
1.0.x: ModbusRTUMaster_0
1.0.0.x: ModbusRequest_0
config.resource1.instance0
mdb_read (WORD)
local_word (WORD)
mdb_rtu_master
Description:
Class Filter: All
Name Class Type Location Initial Value Option Documentation
1 mdb_read External WORD
2 local_word Local WORD
mdb_read
local_word


Config variables
Project properties
Config
Class Filter: All
Name Class Type Location Initial Value Option Documentation
1 modbus_request_MF_1_0_0_function Global UINT noweb, only description and location can be changed, always retain
2 modbus_request_MF_1_0_0_slave_addre Global UINT noweb, only description and location can be changed, always retain
3 modbus_request_MF_1_0_0_regs_numb Global UINT noweb, only description and location can be changed, always retain
4 modbus_request_MF_1_0_0_start_addre Global UINT noweb, only description and location can be changed, always retain
5 modbus_request_MF_1_0_0_enable Global UINT noweb, only description and location can be changed, always retain
6 modbus_request_MF_1_0_0_error_count Global UINT noweb, only description and location can be changed, always retain
7 modbus_request_MF_1_0_0_success_co Global UINT noweb, only description and location can be changed, always retain
8 mdb_read Global WORD

ражаться в WEB-странице контроллера

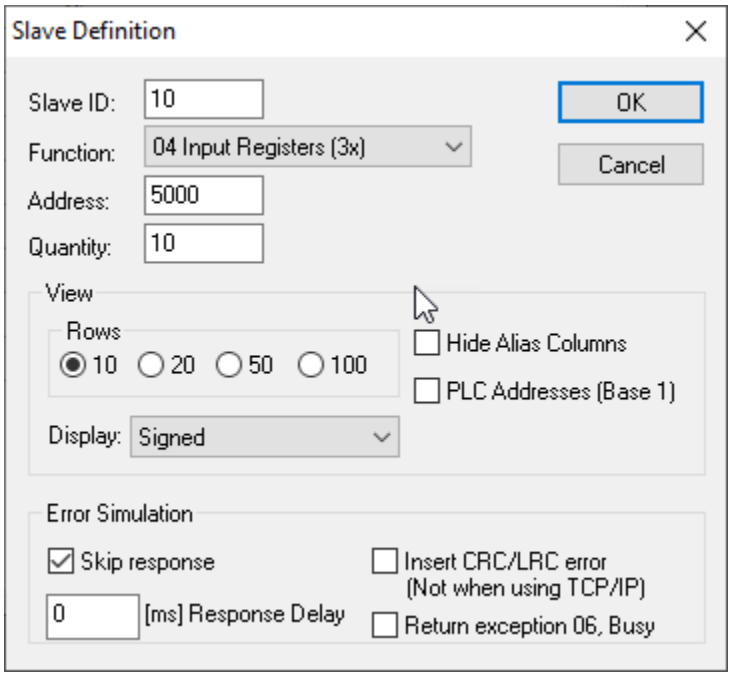
Данные регистры - это ничто иное как параметры ModbusRequest. Если оставить поля пустыми в разделе «Documentation», то есть возможность менять параметры ModbusRequest через WEB-страницу контроллера. Мы с вами так и поступим.

Регистр	Описание
modbus_request_MF_function	Выбор функции
modbus_request_MF_slave_address	Выбор адреса slave-устройства
modbus_request_MF_regs_number	Выбор количества регистров
modbus_request_MF_start_address	Выбор начального адреса
modbus_request_MF_enable	Проверка состояния ModbusRequest
modbus_request_MF_error_counter	Счетчик количества ошибок прием-передачи данных
modbus_request_MF_success_counter	Счетчик количества удачных прием-передачи данных

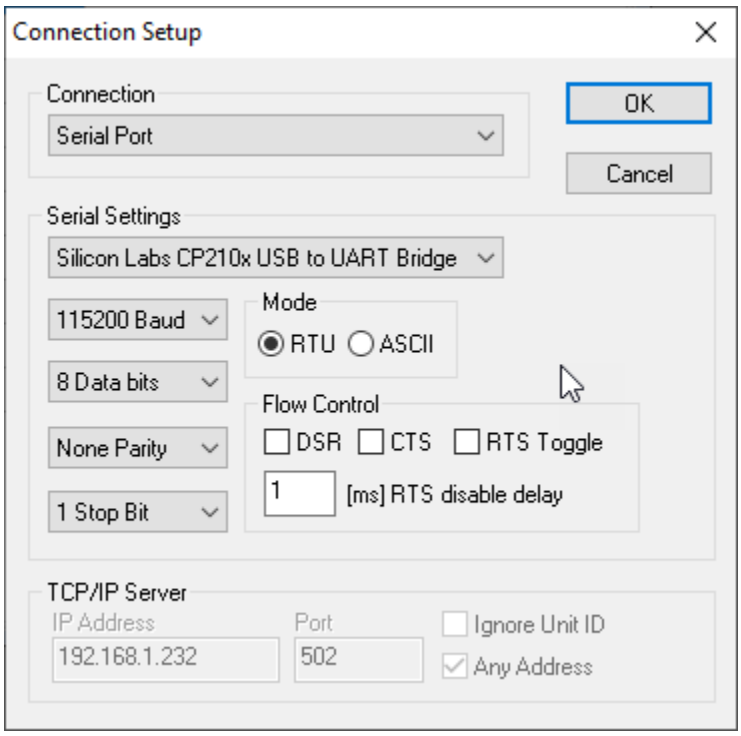
Далее мы назначим Modbus-адрес для переменной *mdb_ read*. Для этого нажимаем по полю «Location» и вводим значение %IW1.0.0.0.

Совет: Для того, чтобы не ошибиться с адресом есть возможность выбора путем нажатия на три точки в «Location».

Наша программа готова, выполняем компилляцию и загружаем в ПЛК BRIC. Выполняем подключение преобразователя интерфейсов USB-HART/RS-485. Также запустим приложение Modbus Slave с настройками, представленными ниже.



Далее запускаем автоматическое инкрементирование регистра по адресу 5000.



Заходим в WEB-страницу контроллера, открываем вкладку «User» и видим изменение «MDB_READ» в соответствии с инкрементируемой величиной. Также показаны детали ModbusRequest.

4.4 Добавление и конфигурирование подмодуля MemoryArea

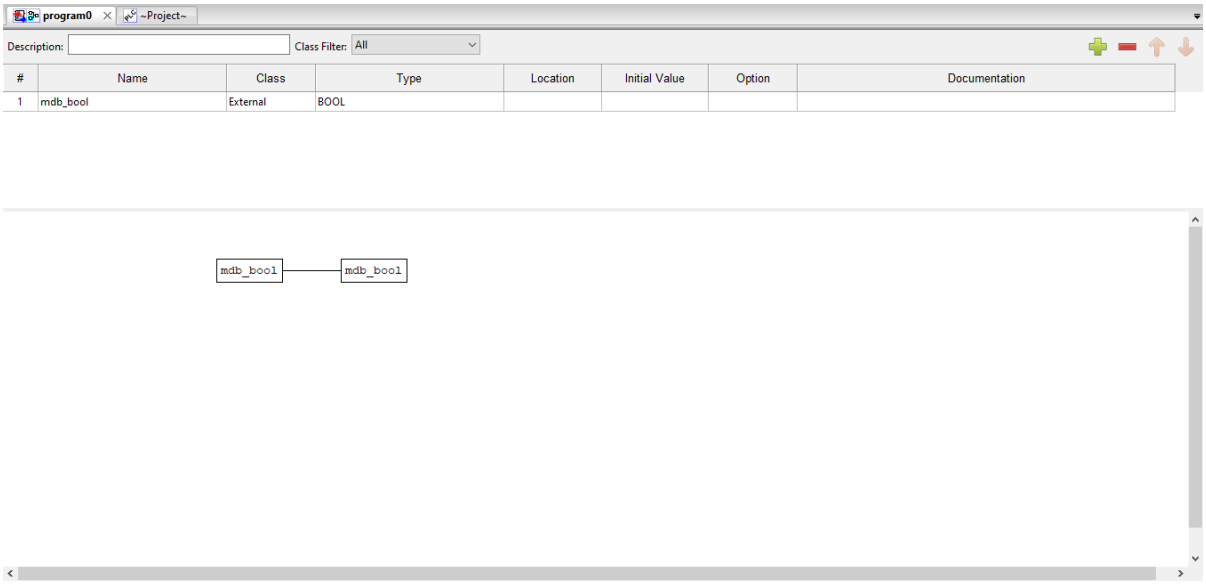
Иногда в каком-либо проекте необходимо увеличить адресное пространство для глобальных переменных. В Beremiz для нашего контроллера BRIC такая возможность имеется. Для того, чтобы к определенным переменным обращались по Modbus необходимо добавить подмодуль MemoryArea. Добавим подмодуль MemoryArea. Рассмотрим подробнее элементы подмодуля.

Таблица 6: Элементы MemoryArea

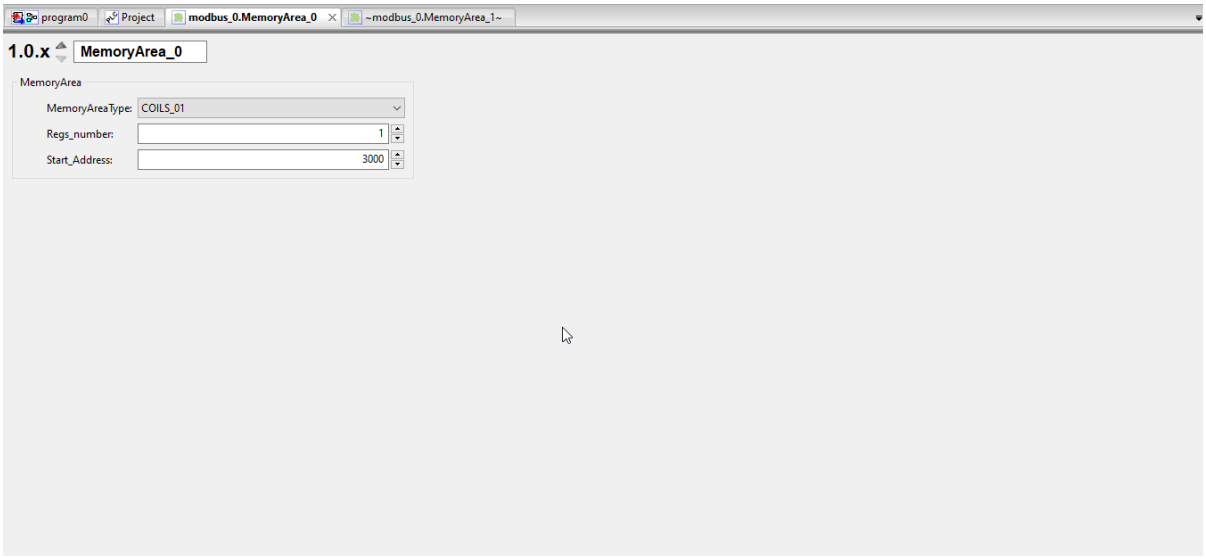
Элемент	Описание
MemoryAreaType	**Выбор функции:** • COILS_01 • HOLDING_REGISTERS_03 • INPUT_DISCRETES_02 • INPUT_REGISTERS_04
Regs_number	Выбор количества регистров
Start_Adress	Выбор стартового адреса

Для нашего урока потребуется программа Modbus Poll, преобразователь usb-rs485 (в нашем случае выбираем преобразователь интерфейсов USB-HART/RS-485). Напишем программу на языке FBD. В

даном случае она будет фиктивной. Добавим переменные mdb_bool и mdb_word с типами данных bool и word. В основной программе выделим только mdb_bool, нам этого достаточно.



Добавляем 2 подмодуля MemoryArea. В первом подмодуле мы будем работать с функцией COILS, а во втором с HOLDING_REGISTERS. Стартовые адреса для каждой переменной запишем собственные, пусть это будет 3000 и 2000.

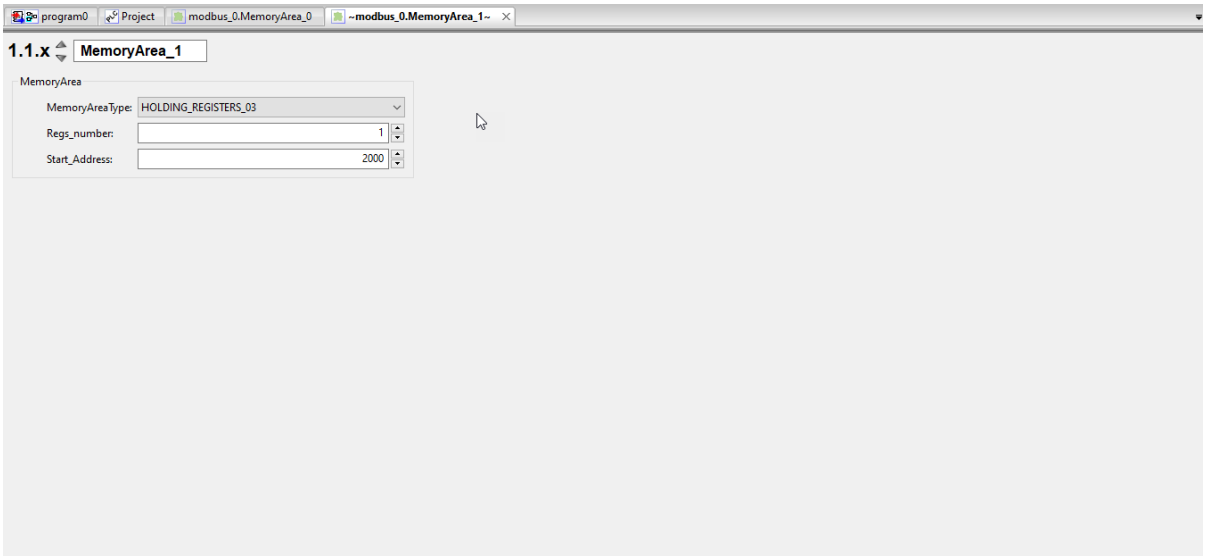


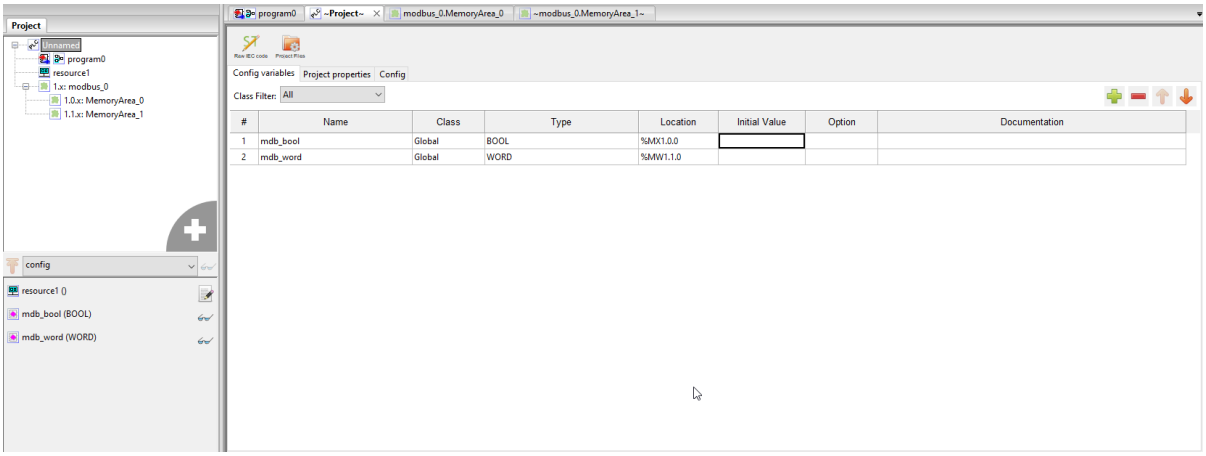
Далее заходим в раздел «Project» и задаем адреса для наших глобальных переменных.

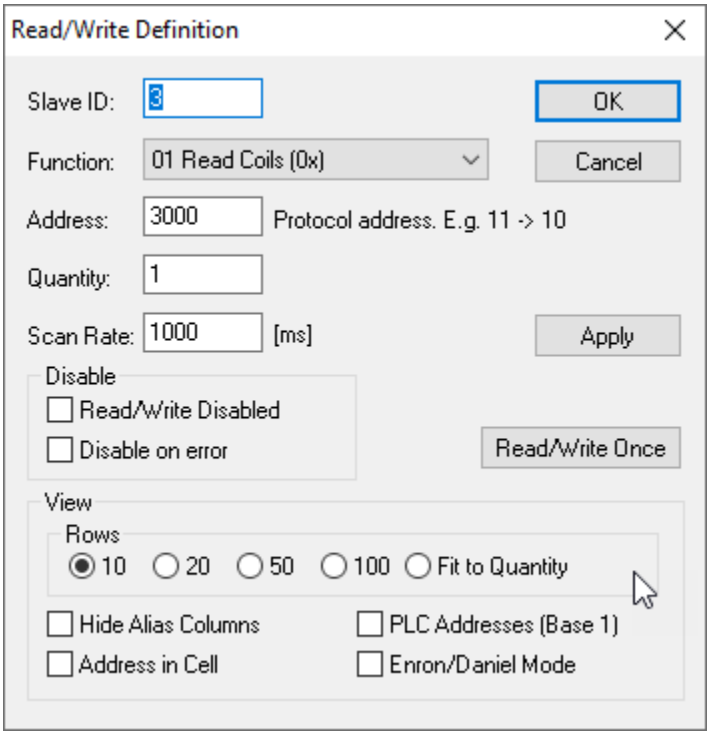
Наша программа готова, делаем компилляцию и загружаем ее в ПЛК BRIC. Подключаемся к контроллеру преобразователем USB-RS485/HART к порту RS_485_1 и запускаем программу Modbus Poll.

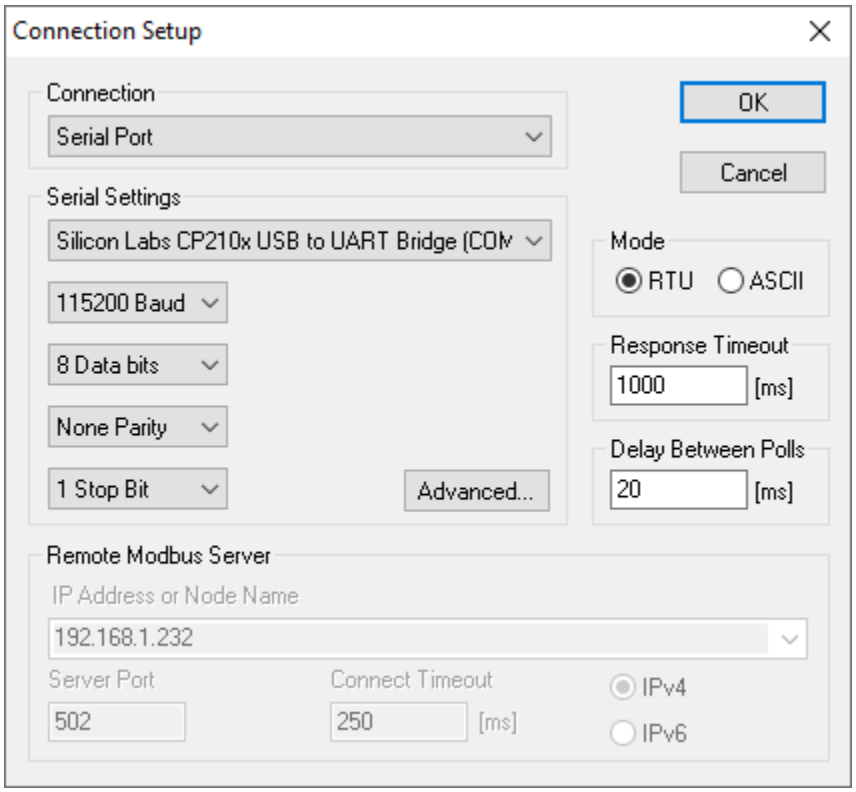
Для начала задаем параметры для чтения/записи глобальной переменной mdb_bool.

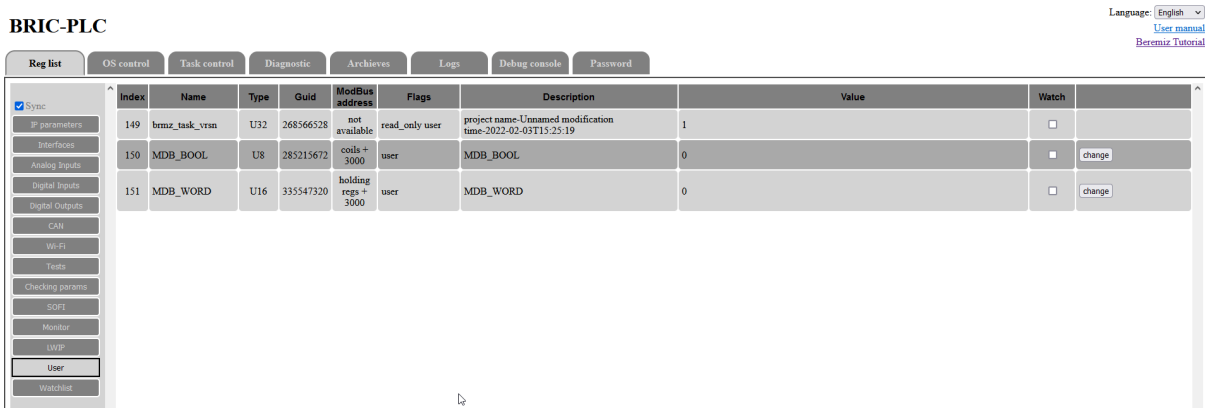
Заходим в WEB-страницу ПЛК и открываем вкладку «User».


program0
Project
modbus_0.MemoryArea_0
~modbus_0.MemoryArea_1~
1.1.x
MemoryArea_1
MemoryArea
MemoryAreaType:
HOLDING_REGISTERS_03
Regs_number:
1
Start_Address:
2000


Project
Unnamed
program0
resource1
1.x: modbus_0
1.0.x: MemoryArea_0
1.1.x: MemoryArea_1
config
resource1 ()
mdb_bool (BOOL)
mdb_word (WORD)
program0
~Project~
modbus_0.MemoryArea_0
~modbus_0.MemoryArea_1~
Config variables
Project properties
Config
Class Filter:
All
Name Class Type Location Initial Value Option Documentation
1 mdb_bool Global BOOL %MX1.0.0
2 mdb_word Global WORD %MW1.1.0


Read/Write Definition
Slave ID:
3
OK
Function:
01 Read Coils (0x)
Cancel
Address:
3000
Protocol address. E.g. 11 -> 10
Quantity:
1
Scan Rate:
1000
[ms]
Apply
Disable
Read/Write Disabled
Disable on error
Read/Write Once
View
Rows
10
20
50
100
Fit to Quantity
Hide Alias Columns
PLC Addresses (Base 1)
Address in Cell
Enron/Daniel Mode


Connection Setup
Connection
Serial Port
OK
Cancel
Serial Settings
Silicon Labs CP210x USB to UART Bridge (COM
115200 Baud
8 Data bits
None Parity
1 Stop Bit
Advanced...
Mode
RTU
ASCII
Response Timeout
1000
[ms]
Delay Between Polls
20
[ms]
Remote Modbus Server
IP Address or Node Name
192.168.1.232
Server Port
502
Connect Timeout
250
[ms]
IPv4
IPv6



Итак, связь установлена, начинаем нашу проверку. Заходим в Modbus Poll и меняем значение регистра. Замечаем, что изменилось значение глобального переменной MDB_BOOL.

Проверим обратную связь, изменим значение переменной MDB_BOOL через WEB-страницу.

Примечание: Проверка второй глобальной переменной MDB_WORD остается как самостоятельная работа

4.5 Добавление и конфигурирование подмодуля ModbusRoute

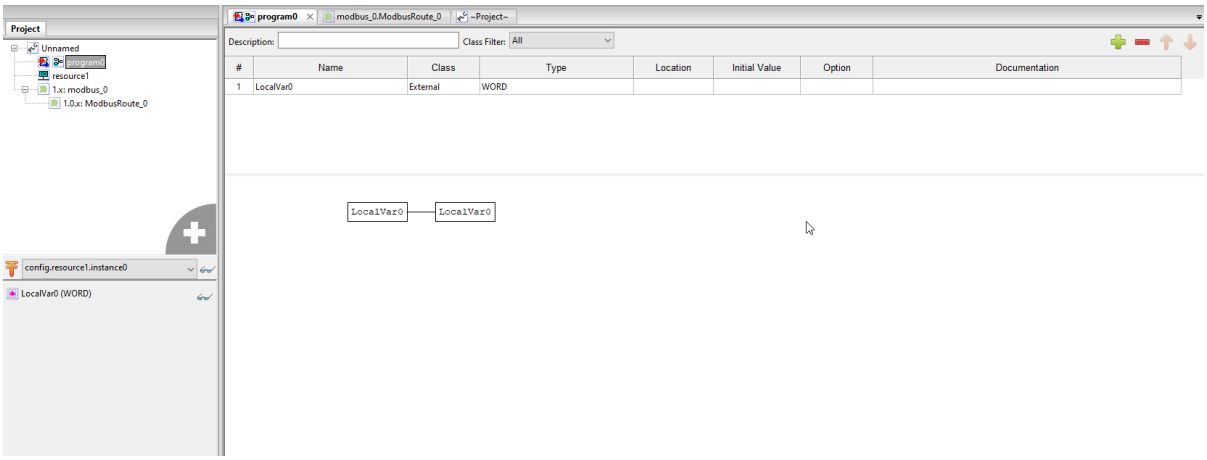
В данном уроке мы с вами разберем тему рентрансляции пакетов с данными из одного канала в другой. Напишем программу, в котором будет налажена приём-передача пакетов Modbus TCP - Modbus RTU.

Рассмотрим элементы подмодуля ModbusRoute.

Таблица 7: Элементы ModbusRoute

Элемент	Описание
ChannelFrom	**Выбор порта передачи:** • PACKET_CHANNEL_TCP (передаваемые данные протоколом TCP через канал связи Ethernet) • PACKET_CHANNEL_UDP (передаваемые данные протоколом UDP через канал связи Ethernet) • RS_232_UART • RS_485_1_UART • RS_485_2_UART • RS_485_IMMO_UART (передаваемые данные протоколом RTU через межмодульный шлейф)
ChannelTo	**Выбор порта приёма:** • RS_232_UART • RS_485_1_UART • RS_485_2_UART • RS_485_IMMO_UART (передаваемые данные протоколом RTU через межмодульный шлейф)
ModbusAdress	Modbus адрес устройства, для которого производится ретрансляция из одного канала в другой[1]

Для нашего урока потребуется программа Modbus Poll и Modbus Slave, преобразователь usb-rs485 (в нашем случае выбираем преобразователь интерфейсов USB-HART/RS-485). Создаем новый проект на любом языке, мы выберем FBD. Добавляем какую-либо глобальную переменную в программе и определим ее во вкладке «Project». Далее добавим подмодуль ModbusRoute.



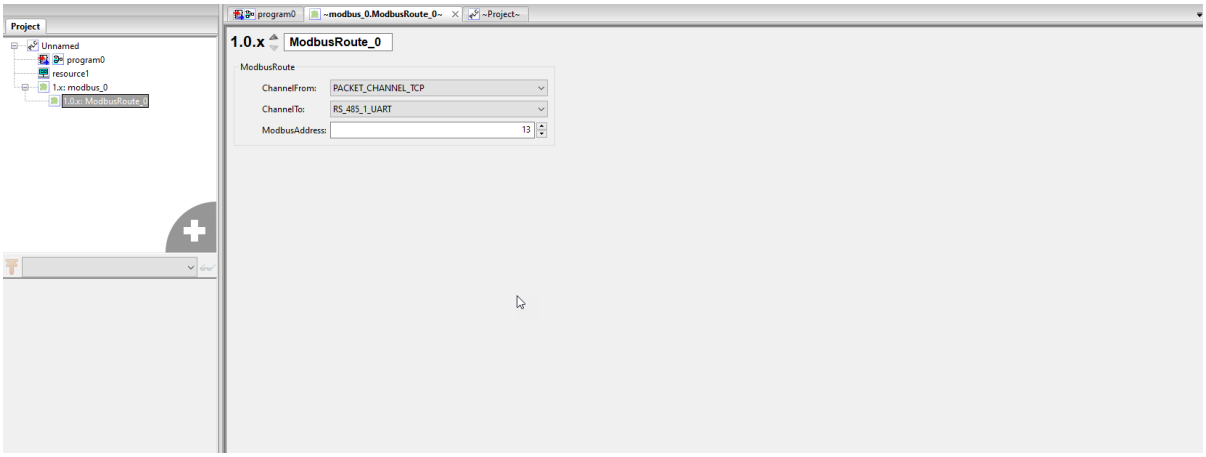
Для нашего случая выберем из TCP в RS485_1 по Modbus-адресу 13.

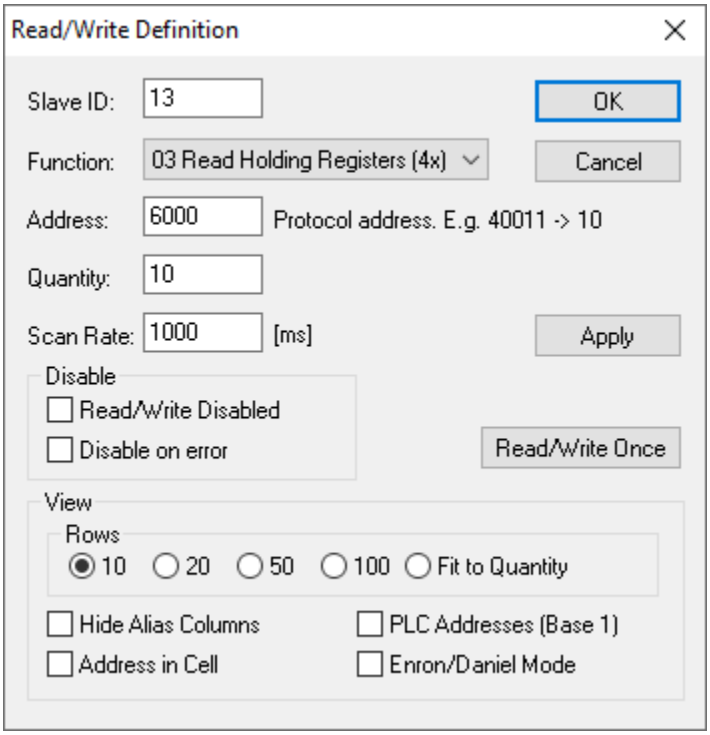
Скомпилируем программу и загружаем в ПЛК. Запускаем программу Modbus Poll, настроим параметры подключения через TCP.

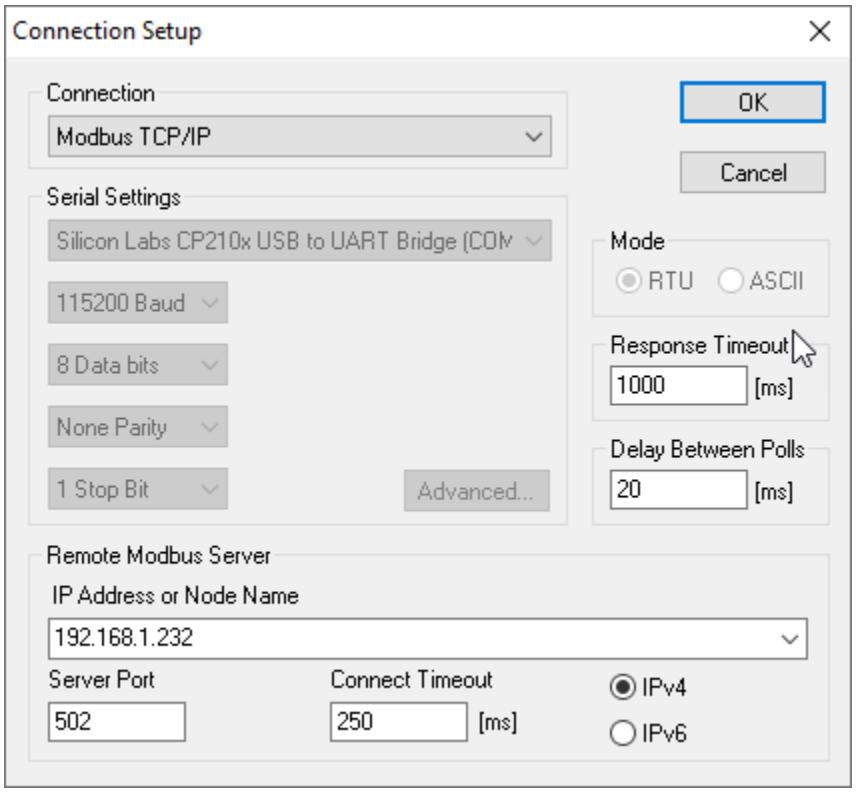
Также параллельно запускаем еще одну программу Modbus Slave, с настройками представленными ниже.

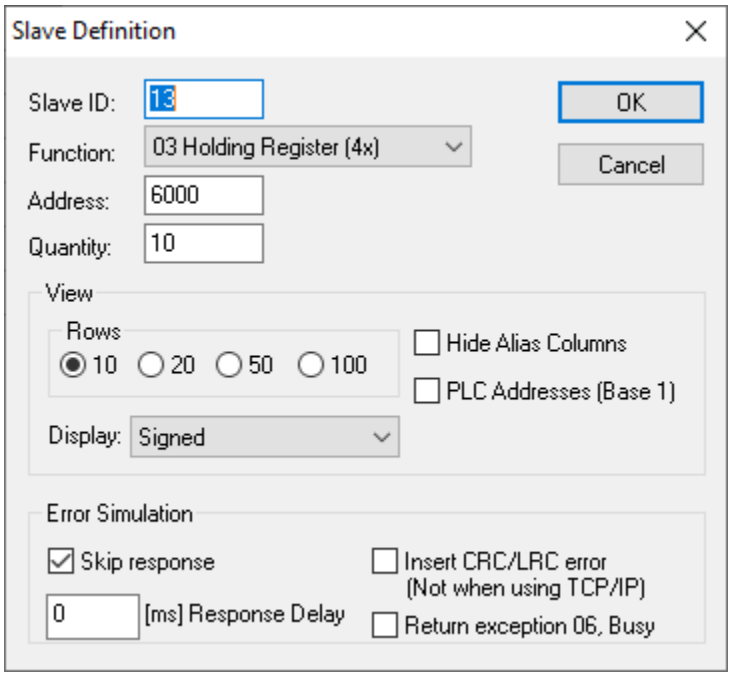
Далее запускаем автоматическое инкрементирование регистра по адресу 6000.

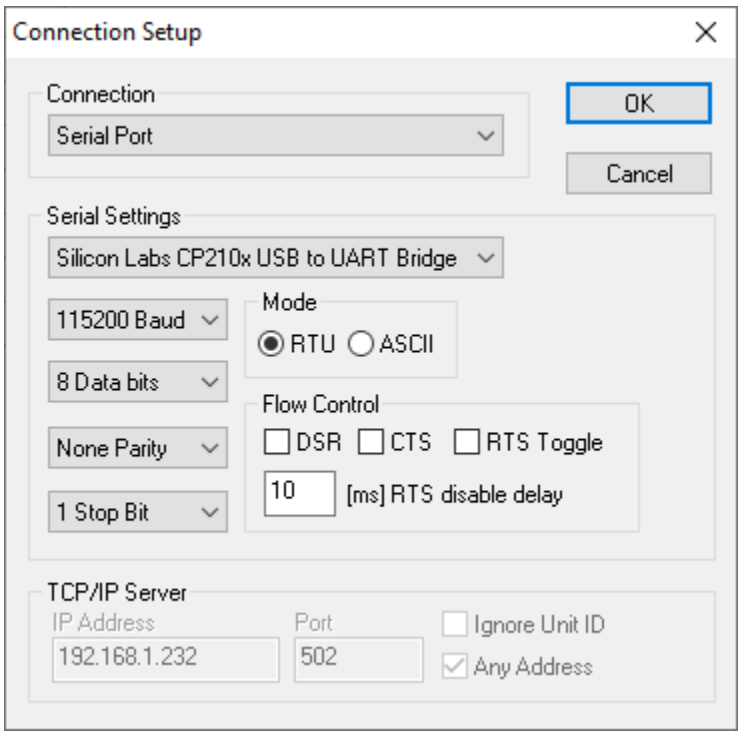
[1] При указании адреса 255 ретранслирует все пакеты полученные с канала «извлечения» в канал «записи».


Project
Unnamed
program0
resource1
1.x: modbus_0
1.0.x: ModbusRoute_0
program0
~modbus_0.ModbusRoute_0~
~Project~
1.0.x
ModbusRoute_0
ModbusRoute
ChannelFrom:
PACKET_CHANNEL_TCP
ChannelTo:
RS_485_1_UART
ModbusAddress:
13


Read/Write Definition
Slave ID:
13
OK
Function:
03 Read Holding Registers (4x)
Cancel
Address:
6000
Protocol address. E.g. 40011 -> 10
Quantity:
10
Scan Rate:
1000
[ms]
Apply
Disable
Read/Write Disabled
Disable on error
Read/Write Once
View
Rows
10
20
50
100
Fit to Quantity
Hide Alias Columns
PLC Addresses (Base 1)
Address in Cell
Enron/Daniel Mode


Connection Setup
Connection
Modbus TCP/IP
OK
Cancel
Serial Settings
Silicon Labs CP210x USB to UART Bridge (COM
115200 Baud
8 Data bits
None Parity
1 Stop Bit
Advanced...
Mode
RTU
ASCII
Response Timeout
1000
[ms]
Delay Between Polls
20
[ms]
Remote Modbus Server
IP Address or Node Name
192.168.1.232
Server Port
502
Connect Timeout
250
[ms]
IPv4
IPv6


Slave Definition
Slave ID:
13
OK
Function:
03 Holding Register (4x)
Cancel
Address:
6000
Quantity:
10
View
Rows
10
20
50
100
Hide Alias Columns
PLC Addresses (Base 1)
Display:
Signed
Error Simulation
Skip response
0
[ms] Response Delay
Insert CRC/LRC error
(Not when using TCP/IP)
Return exception 06, Busy



Как видно, данные ретранслируются из одного канала в другой.

4.6 Чтение и запись данных по протоколу Modbus TCP/IP

Как было видно по предыдущему уроку, у ПЛК BRIC имеется возможность опроса регистров через Modbus TCP. В данном случае контроллер выступает в роли slave-устройства. Напишем программу на языке ST. Добавим переменную mdb_var с типом данных *WORD*.

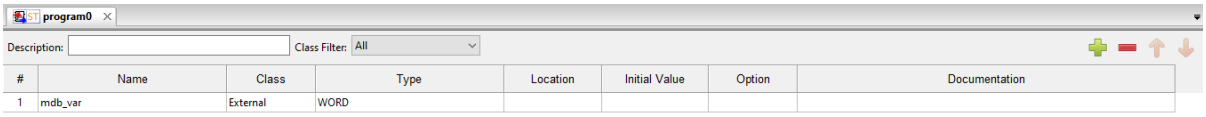
Далее добавим подмодуль MemoryArea. Выберем параметры, представленные ниже на рисунке.

Определим Modbus-адрес для нашей переменной. Заходим в раздел «Project» и в столбце «Location» прописываем форму «%MW1.0.0».

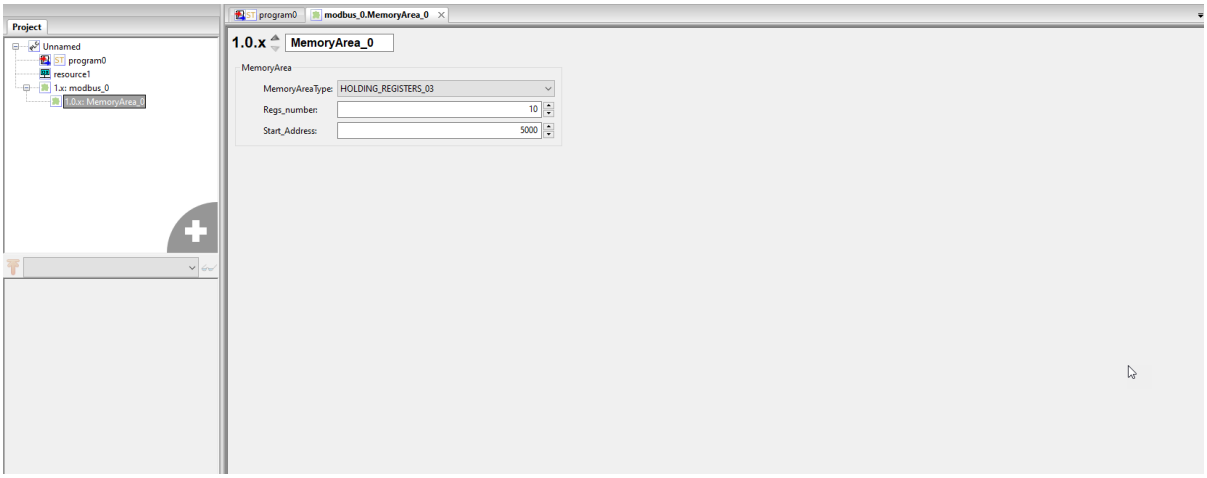
Скомпилируем и заргужаем программу в ПЛК. Подключаемся через Ethernet и запускаем программу Modbus Poll. Задаем настройки, представленные ниже.

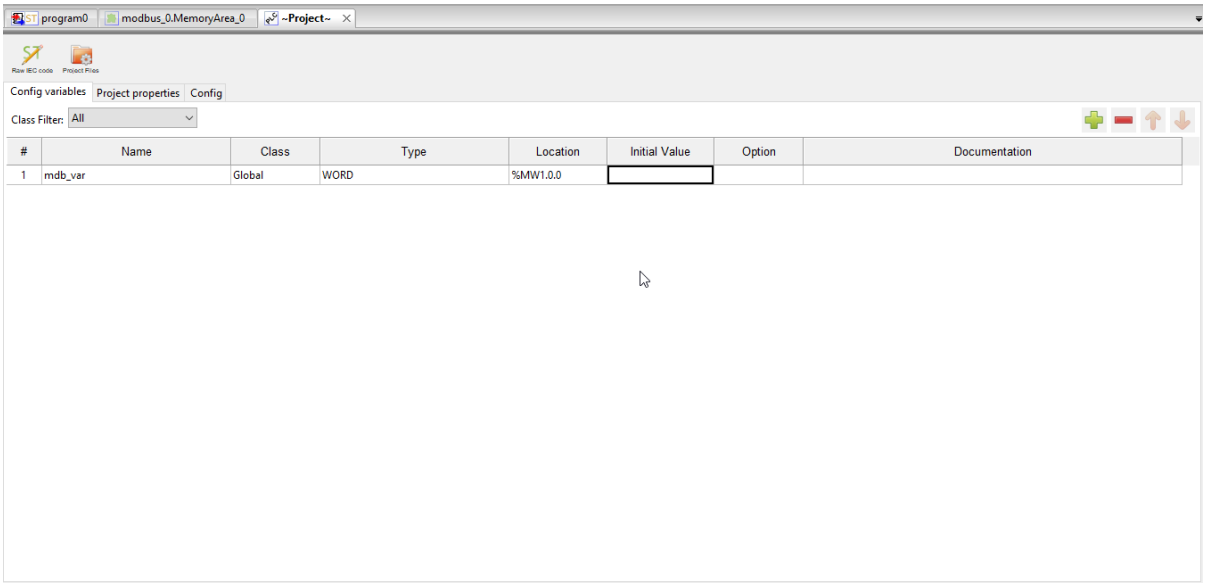
Также заходим в WEB-страницу контроллера (по умолчанию http://192.168.1.232) и заходим во вкладку «User». Далее проверим чтение/запись данных через Modbus TCP.

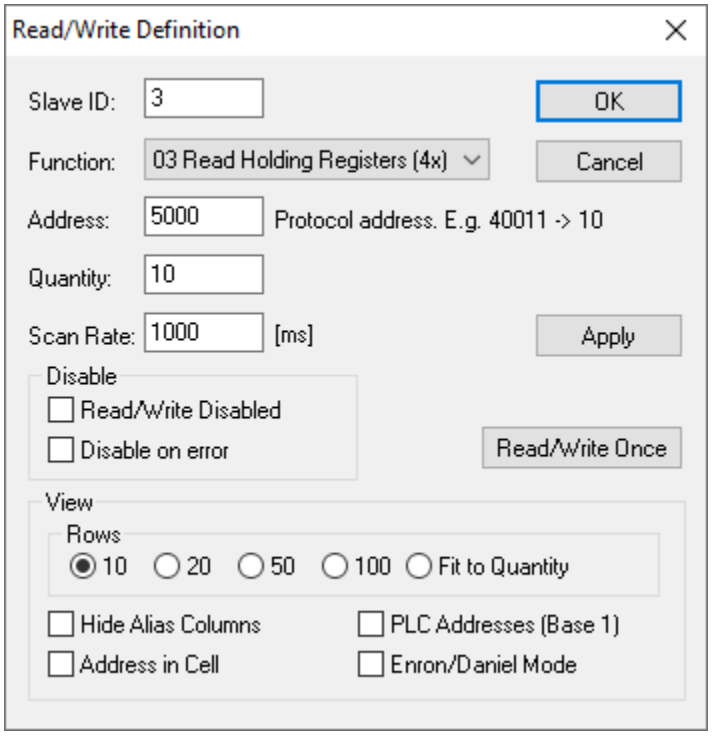
Примечание: Чтение и запись параметров ПЛК через Modbus TCP остается как самостоятельная работа

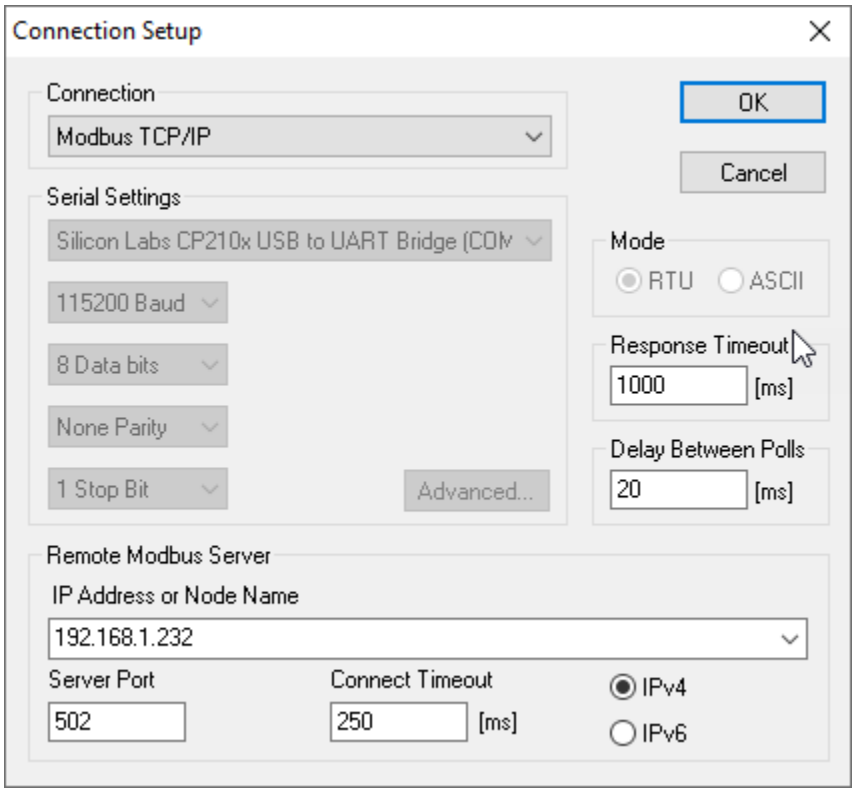

program0
Description:
Class Filter: All
Name Class Type Location Initial Value Option Documentation
1 mdb_var External WORD


1 mdb_var := mdb_var;


Project
Unnamed
program0
resource1
1.x: modbus_0
1.0.x: MemoryArea_0
program0
modbus_0.MemoryArea_0
1.0.x
MemoryArea_0
MemoryArea
MemoryAreaType: HOLDING_REGISTERS_03
Regs_number: 10
Start_Address: 5000


program0
modbus_0.MemoryArea_0
~Project~
Raw IEC code
Project Files
Config variables
Project properties
Config
Class Filter: All
Name Class Type Location Initial Value Option Documentation
1 mdb_var Global WORD %MW1.0.0


Read/Write Definition
Slave ID: 3
OK
Function: 03 Read Holding Registers (4x)
Cancel
Address: 5000
Protocol address. E.g. 40011 -> 10
Quantity: 10
Scan Rate: 1000 [ms]
Apply
Disable
Read/Write Disabled
Disable on error
Read/Write Once
View
Rows
10 20 50 100 Fit to Quantity
Hide Alias Columns
PLC Addresses (Base 1)
Address in Cell
Enron/Daniel Mode


Connection Setup
Connection
Modbus TCP/IP
OK
Cancel
Serial Settings
Silicon Labs CP210x USB to UART Bridge (COM
115200 Baud
8 Data bits
None Parity
1 Stop Bit
Advanced...
Mode
RTU ASCII
Response Timeout
1000 [ms]
Delay Between Polls
20 [ms]
Remote Modbus Server
IP Address or Node Name
192.168.1.232
Server Port
502
Connect Timeout
250 [ms]
IPv4
IPv6

Глава 5

Архивы

5.1 Создание архива. Структура архива. Добавление пользовательских переменных. Расчет глубины архивации

Архивы являются неотъемлемой частью большинства проектов. В Beremiz имеется возможность создания архивов с добавлением пользовательских переменных. Рассмотрим более подробно данный процесс.

Для добавления в структуру проекта необходимо щелкнуть правой кнопкой мыши в область дерева проекта и нажать «Archive support». Далее необходимо подвести курсор к созданной ветке «archive», щелкнуть левой клавишей мыши и выбрать пункт «Add Arch», как показано ниже:

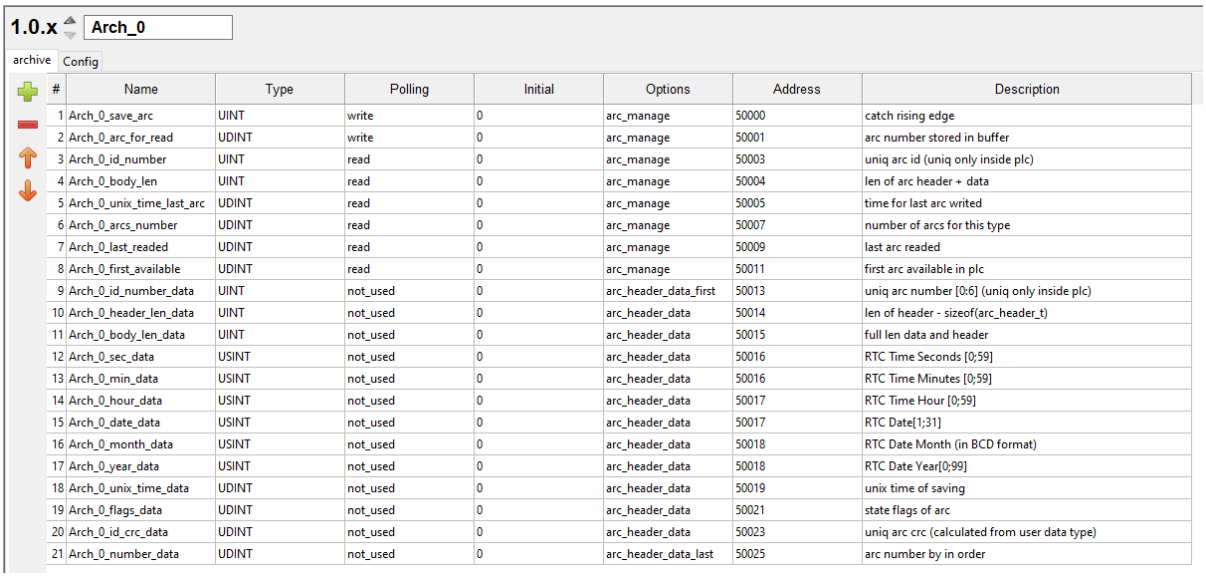
Рис. 1: *Добавление модуля архива в проект*

Рассмотрим более детально окно конфигурации модуля архива.

Таблица 1: Описание archive_manage

Имя	Тип данных	Описание
Arch_save_arc	UINT	Триггер для записи данных в архив: при изменении из 0 в 1 данные записываются
Arch_arc_for_read	UDINT	Данные архива из буфера, представленные для чтения
Arch_id_number	UINT	Уникальный идентификатор архива
Arch_body_len	UINT	Размер архива с учетом заголовка и пользовательских данных
Arch_unix_time_last_arc	UDINT	Время записи последних данных в архив в формате UNIX
Arch_arcs_number	UDINT	Количество архивов, доступных для чтения
Arch_last_readed	UDINT	Номер последнего прочитанного архива
Arch_first_available	UDINT	Номер первого архива, доступного для чтения

Последующие данные являются структурой самого архива.



#	Name	Type	Polling	Initial	Options	Address	Description
1	Arch_0_save_arc	UINT	write	0	arc_manage	50000	catch rising edge
2	Arch_0_arc_for_read	UDINT	write	0	arc_manage	50001	arc number stored in buffer
3	Arch_0_id_number	UINT	read	0	arc_manage	50003	uniq arc id (uniq only inside plc)
4	Arch_0_body_len	UINT	read	0	arc_manage	50004	len of arc header + data
5	Arch_0_unix_time_last_arc	UDINT	read	0	arc_manage	50005	time for last arc writed
6	Arch_0_arcs_number	UDINT	read	0	arc_manage	50007	number of arcs for this type
7	Arch_0_last_readed	UDINT	read	0	arc_manage	50009	last arc readed
8	Arch_0_first_available	UDINT	read	0	arc_manage	50011	first arc available in plc
9	Arch_0_id_number_data	UINT	not_used	0	arc_header_data_first	50013	uniq arc number [0:6] (uniq only inside plc)
10	Arch_0_header_len_data	UINT	not_used	0	arc_header_data	50014	len of header - sizeof(arc_header_t)
11	Arch_0_body_len_data	UINT	not_used	0	arc_header_data	50015	full len data and header
12	Arch_0_sec_data	USINT	not_used	0	arc_header_data	50016	RTC Time Seconds [0;59]
13	Arch_0_min_data	USINT	not_used	0	arc_header_data	50016	RTC Time Minutes [0;59]
14	Arch_0_hour_data	USINT	not_used	0	arc_header_data	50017	RTC Time Hour [0;59]
15	Arch_0_date_data	USINT	not_used	0	arc_header_data	50017	RTC Date[1;31]
16	Arch_0_month_data	USINT	not_used	0	arc_header_data	50018	RTC Date Month (in BCD format)
17	Arch_0_year_data	USINT	not_used	0	arc_header_data	50018	RTC Date Year[0;99]
18	Arch_0_unix_time_data	UDINT	not_used	0	arc_header_data	50019	unix time of saving
19	Arch_0_flags_data	UDINT	not_used	0	arc_header_data	50021	state flags of arc
20	Arch_0_id_crc_data	UDINT	not_used	0	arc_header_data	50023	uniq arc crc (calculated from user data type)
21	Arch_0_number_data	UDINT	not_used	0	arc_header_data_last	50025	arc number by in order

Рис. 2: *Окно параметров архива*

Таблица 2: Описание структуры archive_header_data

Имя	Тип данных	Описание
Arch_id_number_data	UINT	Номер архива данных [0:6]
Arch_header_len_data	UINT	Длина заголовка архива
Arch_body_len_data	UINT	Длина всего архива (header_len + body_len)
Arch_sec_data	USINT	Данные секунд часов реального времени
Arch_min_data	USINT	Данные минут часов реального времени
Arch_hour_data	USINT	Данные часов часов реального времени
Arch_date_data	USINT	Данные дней часов реального времени
Arch_month_data	USINT	Данные месяцев часов реального времени
Arch_year_data	USINT	Данные годов часов реального времени
Arch_date_data	USINT	Данные дней часов реального времени
Arch_unix_time_data	UDINT	Время в формате UNIX
Arch_flags_data	UDINT	Данные о выставленных флагах архива
Arch_id_crc_data	UDINT	Данные об уникальном CRC архива
Arch_number_data	UDINT	Данные о номере архива в списке

Пользовательские данные добавляются при нажатии кнопки  в левом верхнем углу окна, удаляются с помощью кнопки .

Рис. 3: *Добавление и удаление пользовательских данных в архив*

Внимание: Название и тип пользовательских данных должны совпадать с созданными данными программы архива

Размер выделяемой памяти для одного архива обеспечивает минимальную глубину архивации в 1000 записей при максимальном размере пользовательских данных 256 байт. При уменьшении размера пользовательских данных, глубина архивации увеличивается.

Точное значение глубины архивации можно вычислить по формуле:

arc_num = 284000 / (header_len + body_len),

где header_len - длина заголовка (28 байт);

body_len - длина пользовательских данных.

Например, при записи в архив 2-х пользовательских данных типа REAL (по 4 байта) и 2-х данных типа UINT (по 2 байта), то глубина архивации будет составлять:

arc_num = 284000 / (28 + (2 * 4 + 2 * 2)) = 284000 / 40 = 7100 записей.

Допустим, что программа архивации данных записывает 1 раз в 10 минут, то есть 24 * (60 / 10) = 144 записей за сутки. Тогда глубину архивации можно выразить в виде:

7100 / 144 = 49,3 суток.

5.2 Архив с циклической записью данных

В данном уроке ознакомимся с архивированием данных с циклической записью. Напишем 2 программы с разными зависимостями от циклов:

- цикл самой программы (resource)
- цикл от часов реального времени ПЛК (RTC).

Для первого варианта напишем программу на языке ST. Добавим локальную булеву переменную *arc_saver*. Также из окна регистров архива добавим *Arch_save_arc*. В паре с переменной *arc_saver* они будут являться «триггером» записи данных в архив. Программа будет записывать итерацию переменной с циклом в 1 минуту. Программа имеет следующий вид:

Для того, чтобы запись происходила 1 раз в минуту, необходимо задать *task* с интервалом в 30 секунд в ресурсах, так как в программе идет переключение из «0» в «1» за 1 цикл и для обратного переключения необходим дополнительный цикл.

В окне регистров архива «Arch_0» добавим переменную *value1_arch* с типом данных «UINT».

После загрузки программы в ПЛК инкрементируемые данные переменной *value1* будут записываться в архив с периодичностью 1 раз в минуту.

Для второго варианта напишем программу на языке FBD. В данном случае будем привязываться к часам реального времени (RTC). Из «Library» добавим функциональный блок STRUCT_REAL_TIME. Переменную *minute_trigger* определим как кратную 1 минуте. От ее состояния определяется запись данных в архив. Добавим подмодуль «Arch_1». В программу копируем регистр *Arch_1_save_arc*. Также в программу добавим инкрементируюмую переменную *value2* и записываюмую в архив *value2_arch*. Конечный вариант программы показан на рисунке ниже:

В ресурсах добавляем *task* равной 25 мс и добавляем в «Instances».

В окне регистров архива «Arch_1» добавим переменную *value2_arch* с типом данных «UDINT».

После загрузки программы в ПЛК инкрементируемые данные переменной *value2* будут записываться в архив с периодичностью 1 раз в минуту.

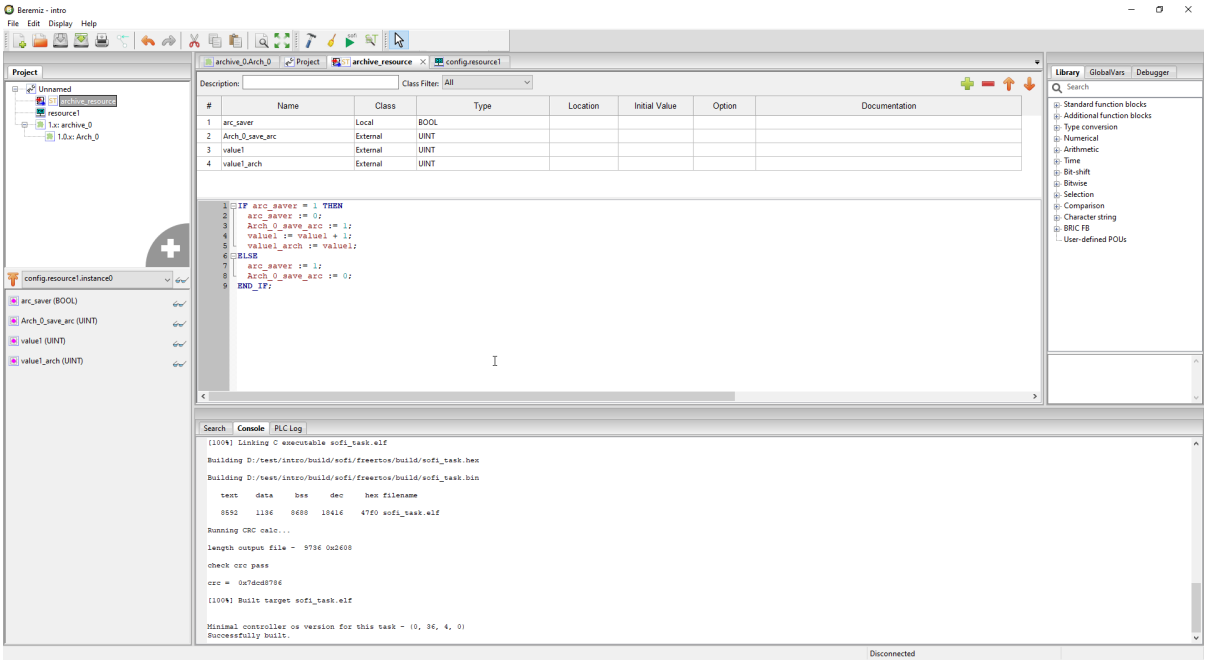


Рис. 4: *Программа archive_resource*

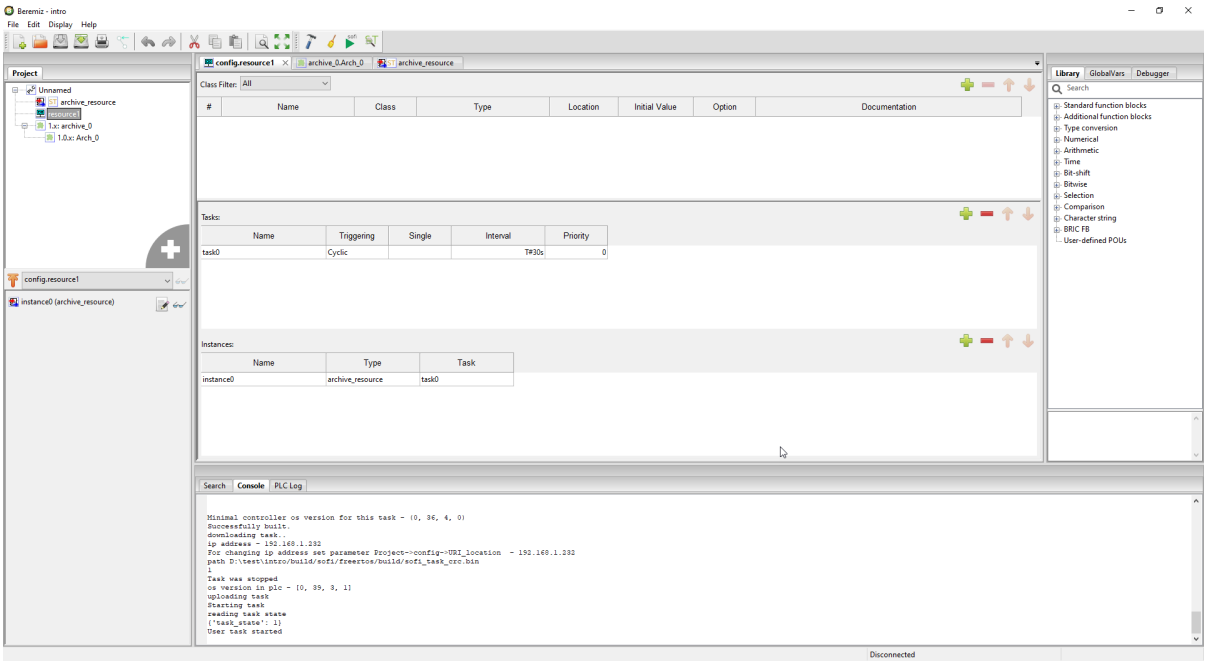


Рис. 5: *Ресурс программы*

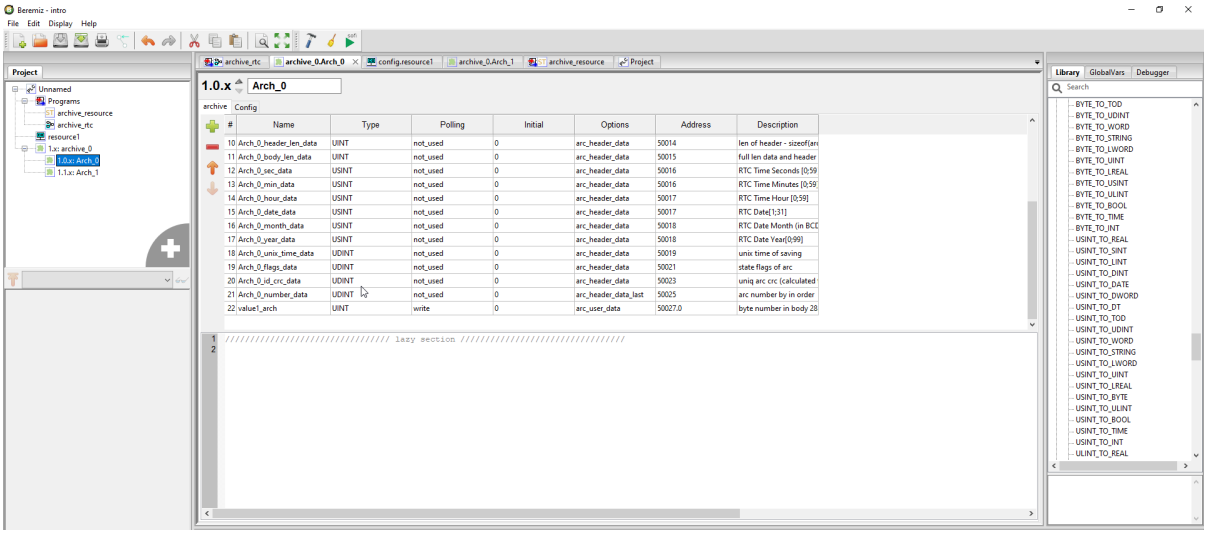


Рис. 6: *Окно регистров архива*

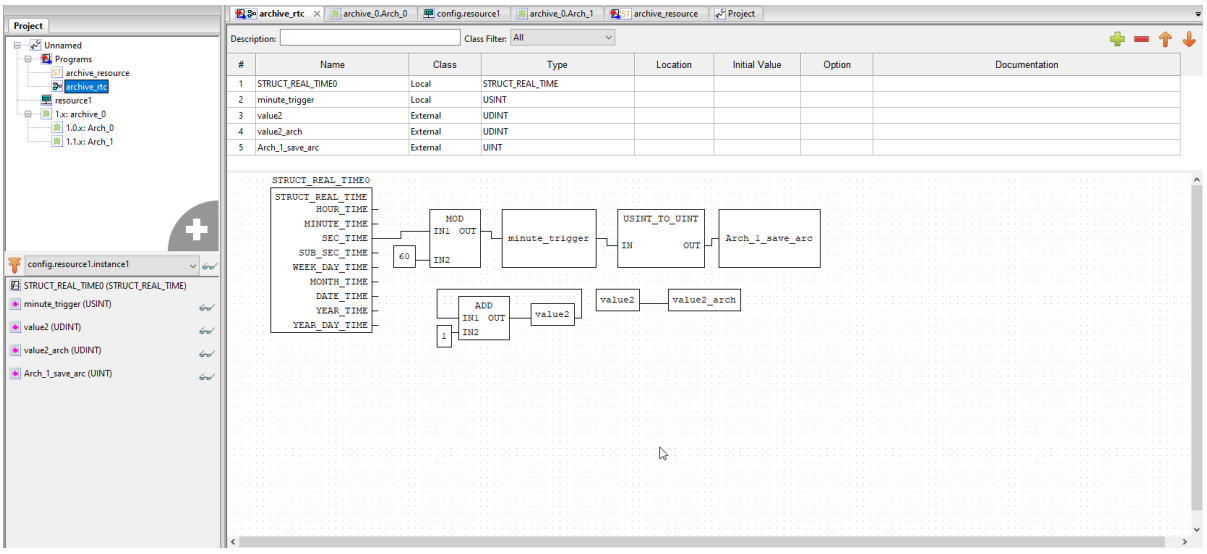


Рис. 7: *Программа archive_rtc*

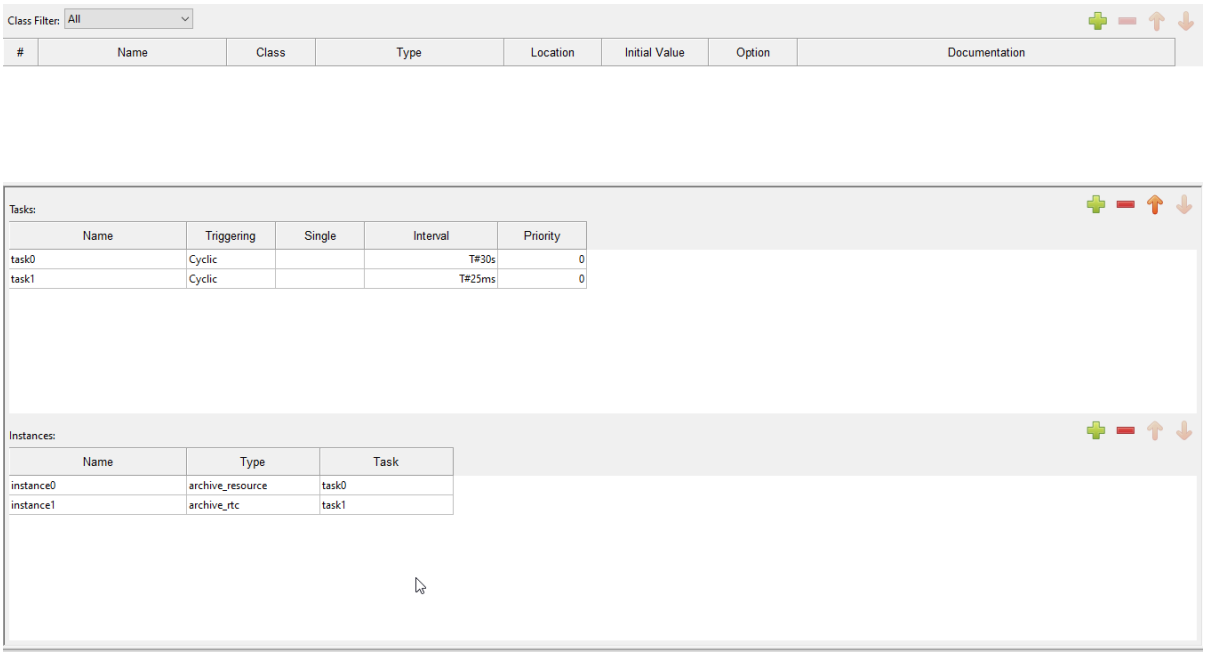


Рис. 8: *Ресурс программы*

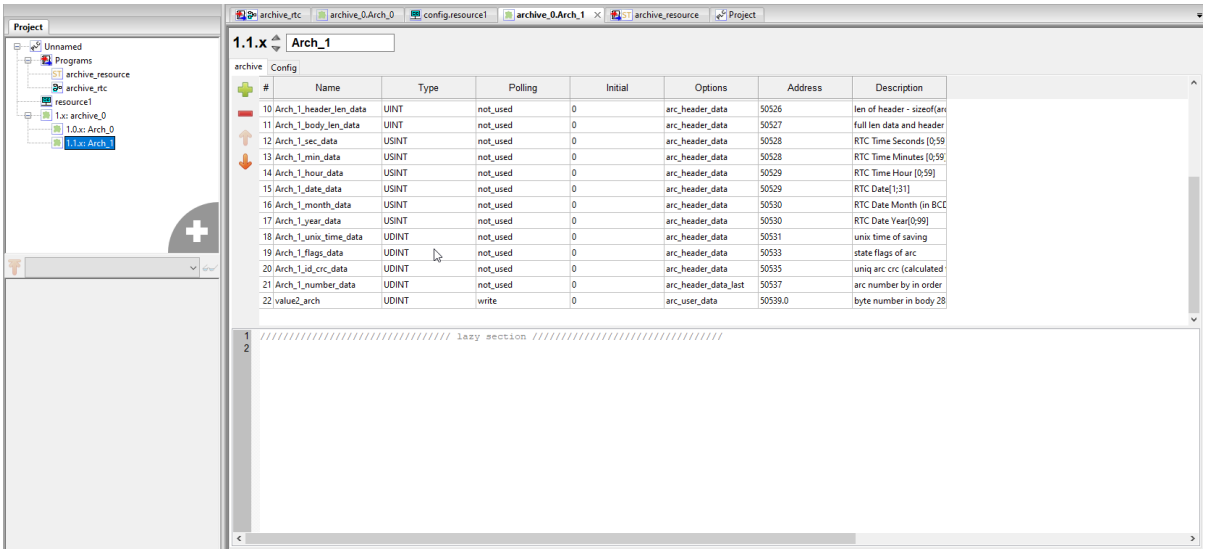


Рис. 9: *Окно регистров архива*

5.3 Архив с записью данных по событию

Для данного урока напишем основную программу «main» на языке FBD и программу для архива «archive» на языке ST. В «main» добавим переменную *value* с типом данных «USINT». Сделаем *value* инкрементируемой. Тип данных «USINT» имеет диапазон от 0 до 255. Добавим булеву переменную *trigger* от значения которой будет зависеть запись данных в архив. Для примера, сделаем так, чтобы в архив записывались данные при *value* равной 60, либо 120, 180, 240. Реализация данного примера представлена ниже.

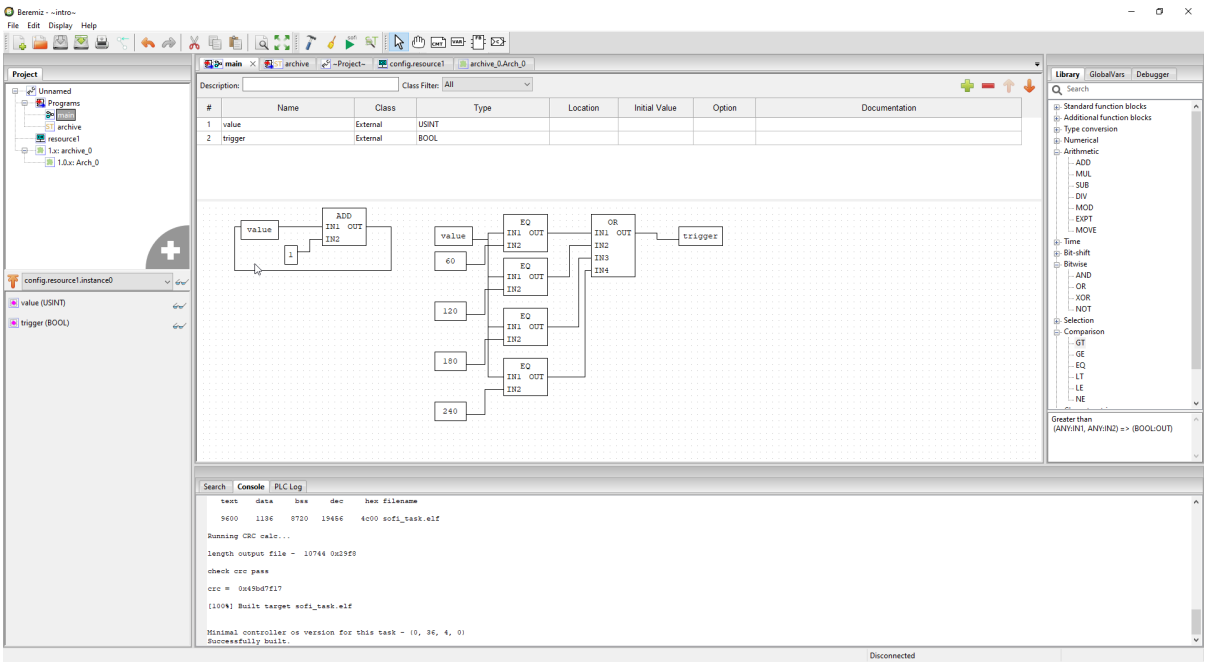


Рис. 10: *Программа «main»*

Примечание: Для того чтобы добавить несколько входов в функциональный блок «OR», необходимо дважды щелкнуть по нему и выбрать количество входов.

В программе «archive» добавим переменные *trigger*, *value*, и регистр *Arch_ save_ arc*. Также создаем переменную *value_ arch*, в которую будут записываться данные *value* при срабатывании *trigger*. Конечный результат программы показан на рисунке ниже:

Далее добавляем переменную *value_ arch* в окно регистров архива. Тип данных указываем как «USINT».

В ресурсах выставим настройки для каждой программы отдельно:

В глобальных переменных необходимо добавить *value* и *trigger*. Для переменной *value* пропишем начальное значение 1, сделаем его «Retain», что означает сохранение значения при возможных перезагрузках или выключениях ПЛК.

После загрузки проекта в ПЛК данные *value* будут записываться в архив при значениях 60, 120, 180 и 240.

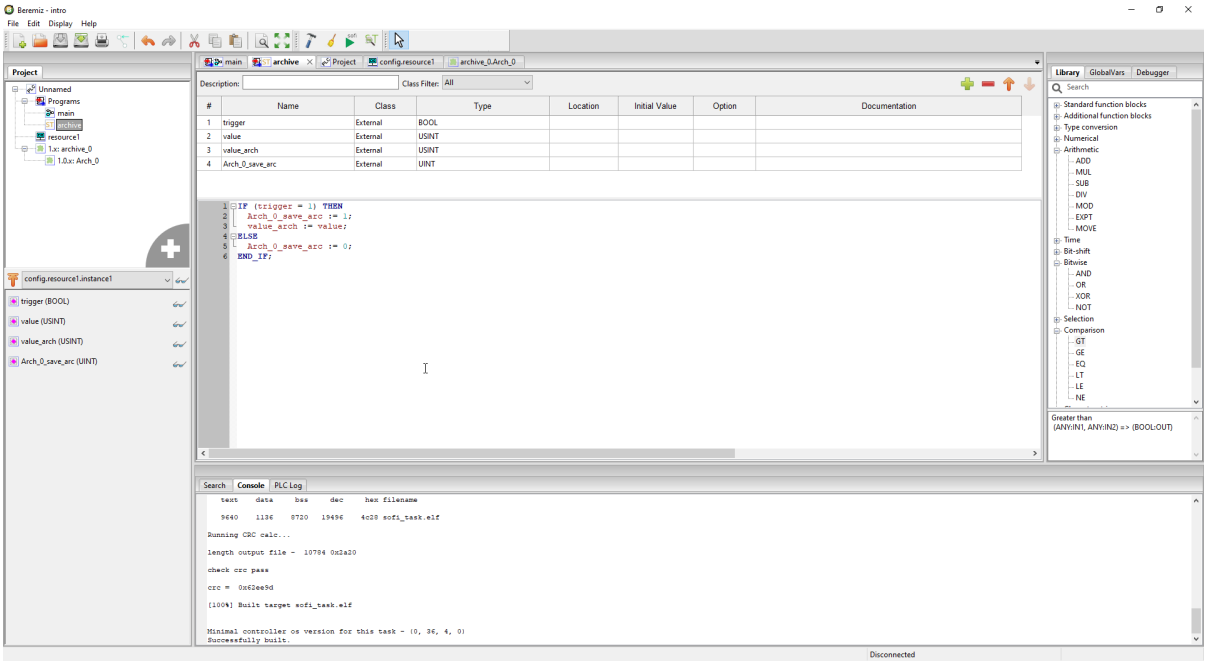


Рис. 11: *Программа «archive»*

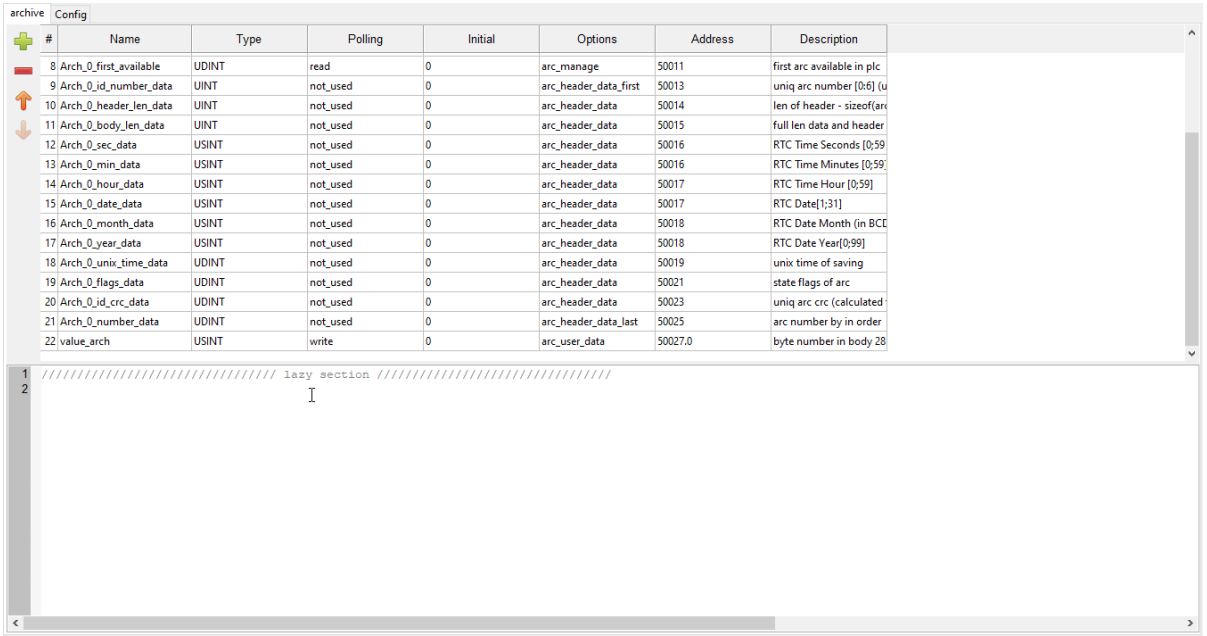


Рис. 12: *Окно регистров архива*

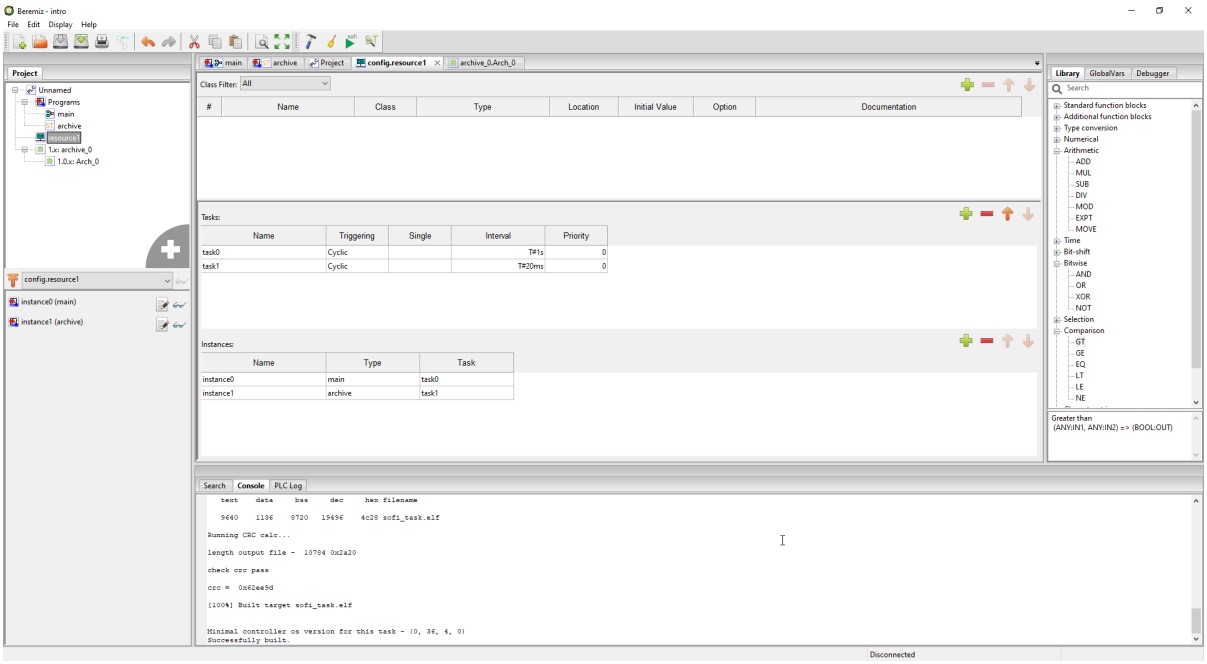


Рис. 13: *Ресурсы программ*

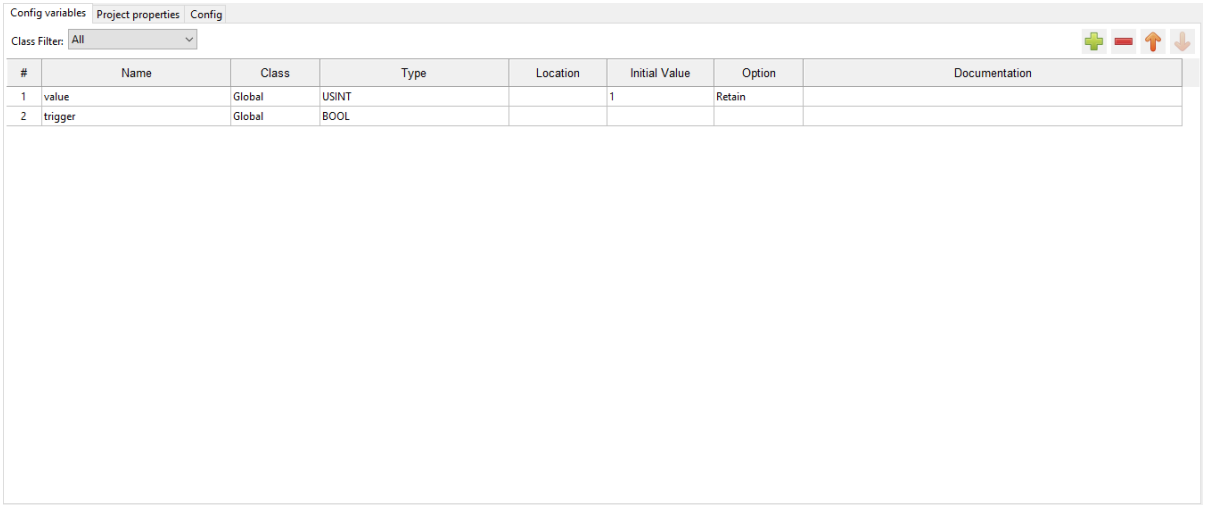


Рис. 14: *Глобальные переменные программ*

5.4 Чтение архивов. WEB - страница «Archieves»

В ПЛК BRIC реализована функция работы с архивами, которые создаем в своей программе. Для того, чтобы начать работу с архивами необходимо открыть вкладку «Archieves». Если в пользовательской программе архивы отсутствуют, то отобразится сообщение «No available archieves».

Напишем основную программу на языке FBD. Добавим инкрементируемую переменную *value_x* , и переменную *value_y*. Для данного урока реализуем программу, описывающую функцию $value\_y = \sin(value\_x)$. В ИСР Beremiz переменные представлены как радианы. Для перевода в градусы воспользуемся формулой:

$$1 = 1 * /180$$

Финальная версия программы представлена на рисунке ниже:

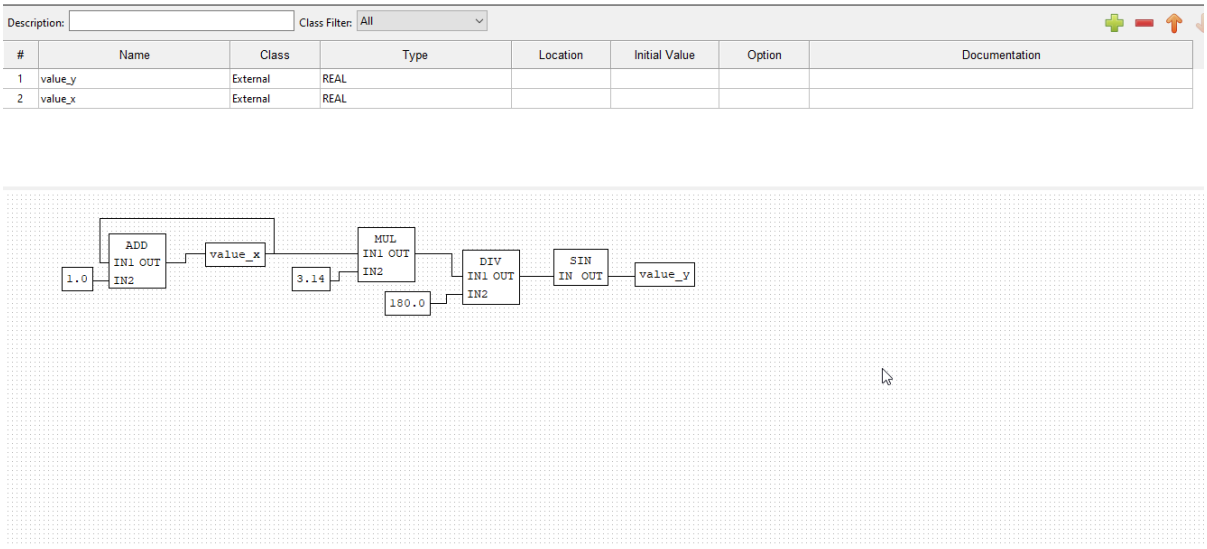


Рис. 15: *Программа «sinus»*

Напишем программу для архива на языке ST. Для начала добавим модуль архивов, оттуда копируем регистр *Arch_save_arc*. Также создаем локальную переменную *trigger*, который будет являться триггером записи данных в архив, переменные *y_ar* и *x_ar*, в которые будут записываться данные от *value_y* и *value_x*.

Финальная версия программы представлена на рисунке ниже:

Добавим наши архивные переменные в окно регистров «archive_0.Arch_0».

Сделаем цикл каждой программы равным 5 секундам. Данные будут записываться в архив каждые 10 секунд.

После загрузки нашей пользовательской программы, заходим в WEB-страницу контроллера, открываем вкладку «Archieves». Таблица «Available archieves» показывает информацию по имеющимся в контроллере архивам:

- «Arch_ID» - идентификатор или номер архива. В нашем случае у нас только 1 архив под номером 0;
- «First» - номер первой доступной записи архива;
- «Last» - номер последней записи архива.

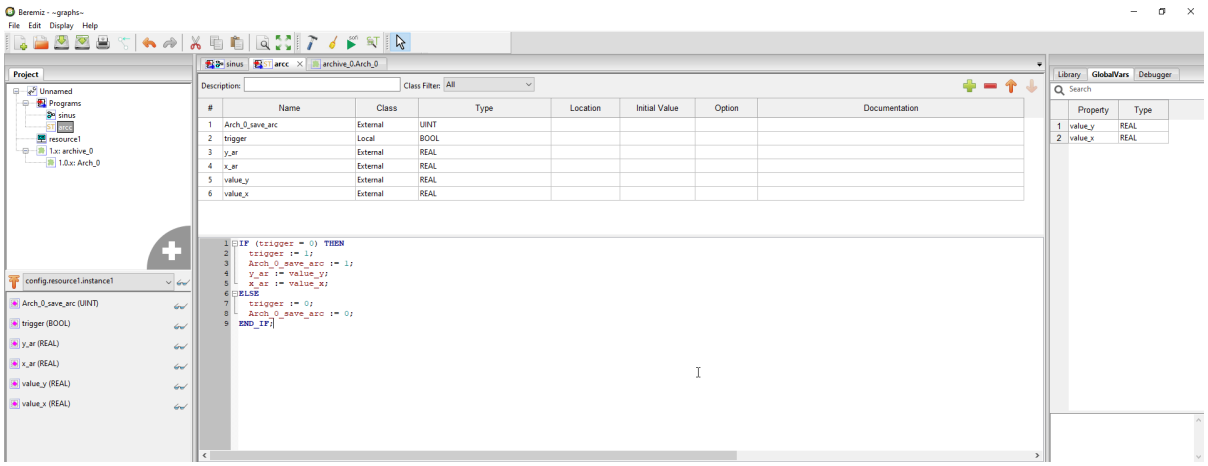
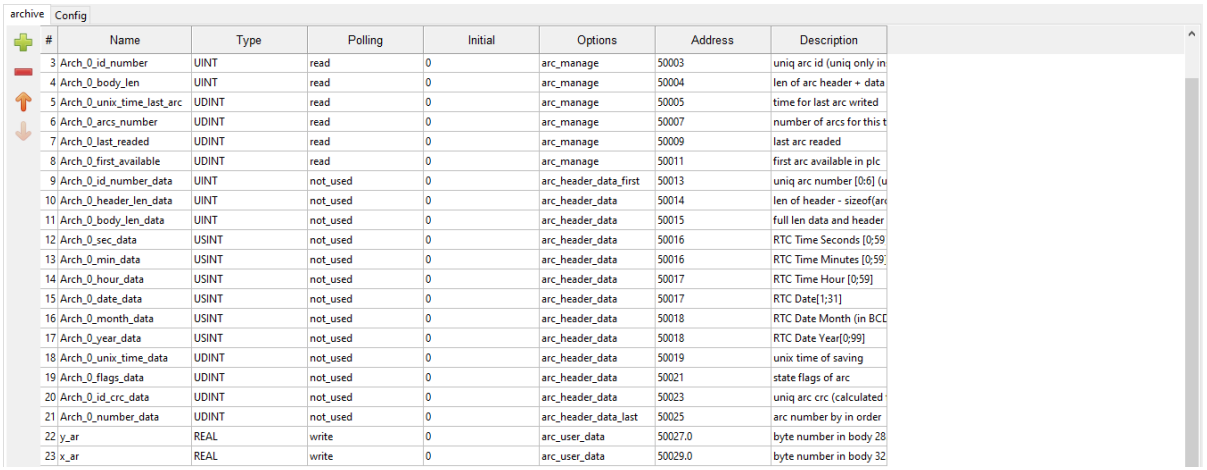


Рис. 16: *Программа «arcc»*



#	Name	Type	Polling	Initial	Options	Address	Description
3	Arch_0_id_number	UINT	read	0	arc_manage	50003	uniq arc id (uniq only in
4	Arch_0_body_len	UINT	read	0	arc_manage	50004	len of arc header + data
5	Arch_0_unix_time_last_arc	UDINT	read	0	arc_manage	50005	time for last arc writed
6	Arch_0_arcs_number	UDINT	read	0	arc_manage	50007	number of arcs for this t
7	Arch_0_last_readed	UDINT	read	0	arc_manage	50009	last arc readed
8	Arch_0_first_available	UDINT	read	0	arc_manage	50011	first arc available in plc
9	Arch_0_id_number_data	UINT	not_used	0	arc_header_data_first	50013	uniq arc number [0:6] (u
10	Arch_0_header_len_data	UINT	not_used	0	arc_header_data	50014	len of header - sizeof(arc
11	Arch_0_body_len_data	UINT	not_used	0	arc_header_data	50015	full len data and header
12	Arch_0_sec_data	USINT	not_used	0	arc_header_data	50016	RTC Time Seconds [0;59
13	Arch_0_min_data	USINT	not_used	0	arc_header_data	50016	RTC Time Minutes [0;59
14	Arch_0_hour_data	USINT	not_used	0	arc_header_data	50017	RTC Time Hour [0;59]
15	Arch_0_date_data	USINT	not_used	0	arc_header_data	50017	RTC Date[1;31]
16	Arch_0_month_data	USINT	not_used	0	arc_header_data	50018	RTC Date Month (in BCD
17	Arch_0_year_data	USINT	not_used	0	arc_header_data	50018	RTC Date Year[0;99]
18	Arch_0_unix_time_data	UDINT	not_used	0	arc_header_data	50019	unix time of saving
19	Arch_0_flags_data	UDINT	not_used	0	arc_header_data	50021	state flags of arc
20	Arch_0_id_crc_data	UDINT	not_used	0	arc_header_data	50023	uniq arc crc (calculated
21	Arch_0_number_data	UDINT	not_used	0	arc_header_data_last	50025	arc number by in order
22	y_ar	REAL	write	0	arc_user_data	50027.0	byte number in body 28
23	x_ar	REAL	write	0	arc_user_data	50029.0	byte number in body 32

Рис. 17: *Окно «Arch_ 0»*

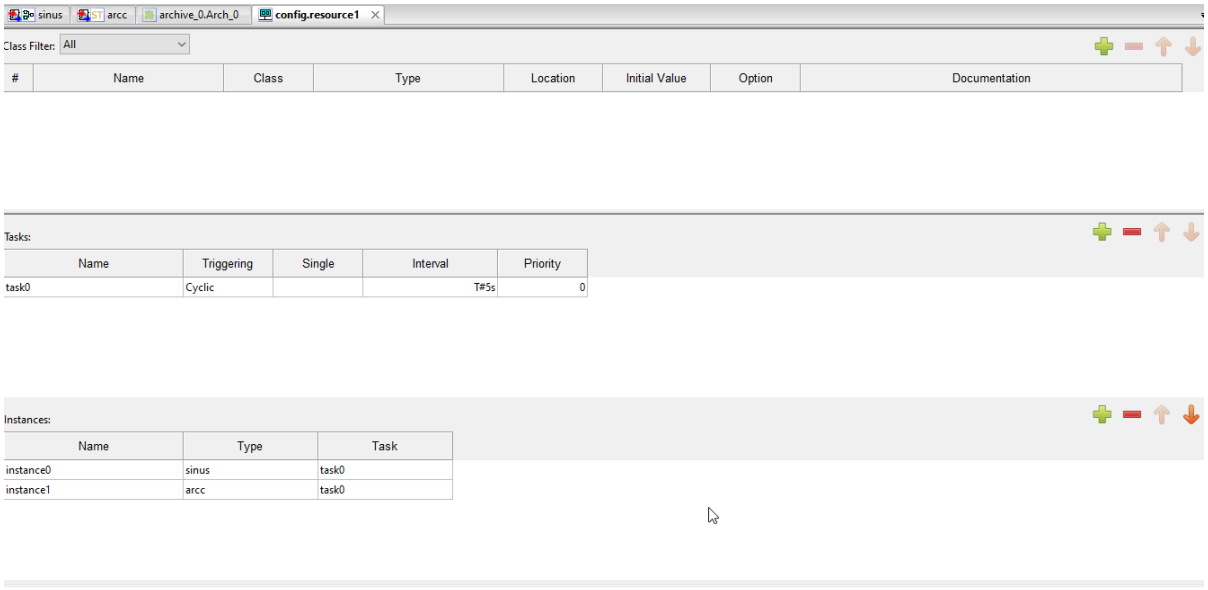


Рис. 18: *Ресурсы программ*

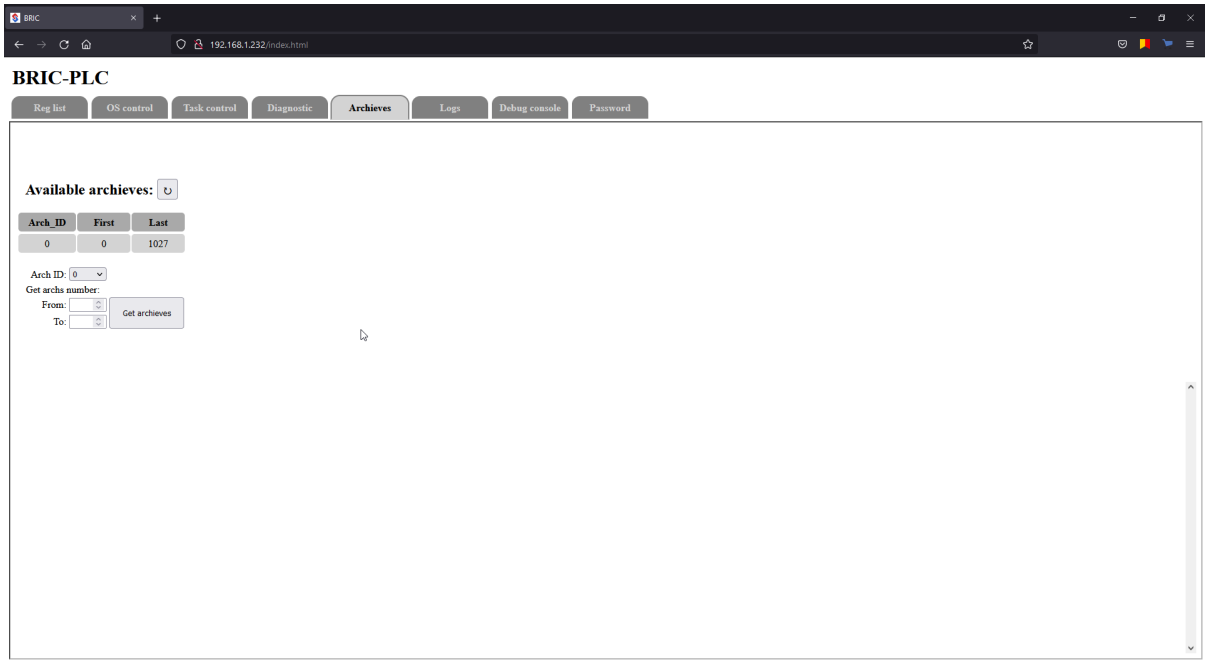


Рис. 19: *Вкладка «Archieves»*

Кнопка с круговой стрелкой «обновить» - для актуализации данных.

В выпадающем списке «Arch_ID» выбираем интересующий нас архив, в нашем случае он один под номером «0»(вообще имеется возможность создания до 10 архивов в одном проекте). Далее укажем интервал записей, которые мы хотим считать в полях «From» и «To». Считывать записи можно в несколько подходов. Давайте получим последние 200 записей.

Рис. 20: *Получение записей*

После того, как получили записи появляется таблица:

- «Archieve header» - это неизменная часть записи, которая содержит:
  - порядковый номер записи (ARCH number);
  - время (Time);
  - дата (Date);
  - дата и время в формате UNIX (Unix);
  - служебные флаги (Flags).
- «Archieve body» - это те параметры, которые мы настраиваем при конфигурации архива в программе. Здесь мы можем видеть наши добавленные переменные *y_ar* и *x_ar*.

Также есть инструменты построения одного или двух графиков. В выпадающем списке «1-st» выберем переменную *y_ar* и нажмем кнопку «Upgrade graph». В результате мы увидим график синусоиды.

Рис. 21: *Получение графика*

Можно поставить галочку «2-nd» и выбрать из выпадающего списка переменную *x_ar*. Теперь можно визуально сопоставить два параметра на одной шкале времени. Ось Y для второго параметра отображается справа.

Рис. 22: *Сопоставление графиков*

Помимо просмотра записей в онлайне, считанный архив можно сохранить в виде файла с расширением «.csv». Кнопка «Save to file» появляется после того, как записи были считаны из контроллера.

5.5 Чтение архивов по «Modbus»

Для модуля архива в ИСР Beremiz по умолчанию задается область Modbus-адресов, начинающаяся с 50000.

Пользовательские переменные по умолчанию начинаются с адреса 50027, и при добавлении новых инкрементируются с учетом типа данных.

Для данного урока напишем программу, через которую будем опрашивать пользовательские переменные модуля архива, а также данные *Archieve header*. Нам понадобится USB-RS485 преобразователь, программное обеспечение для тестирования и наладки устройств и сетей на базе Modbus (Modbus Poll). Напишем программу на языке ST. Добавим переменную *saver*, по изменению которой будет записываться в архив данные переменной *our_value* в виде *arc_our_value*.

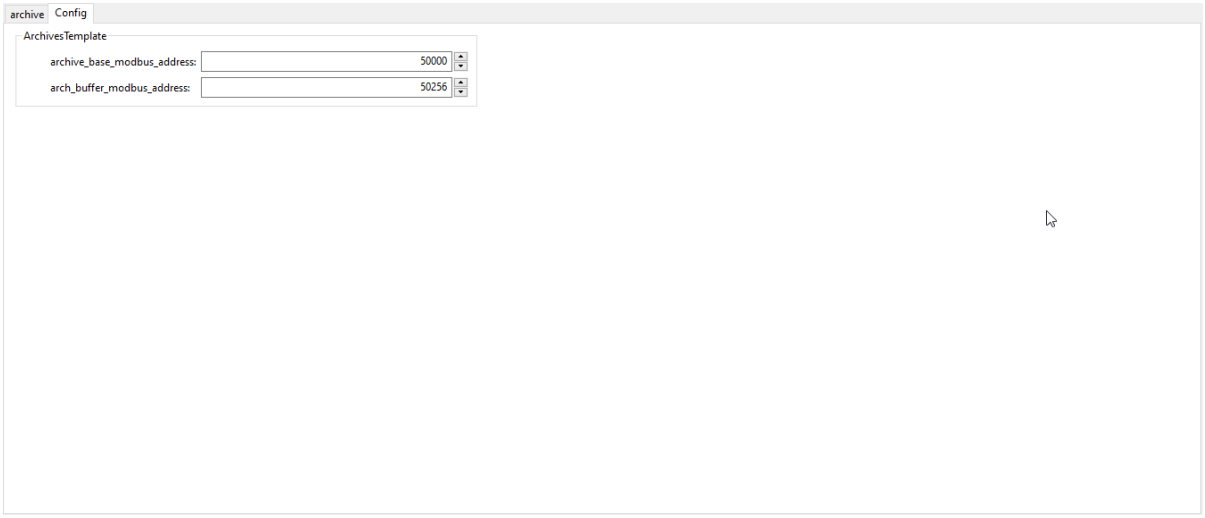


Рис. 23: *Окно конфигурации Modbus-адресов архива*

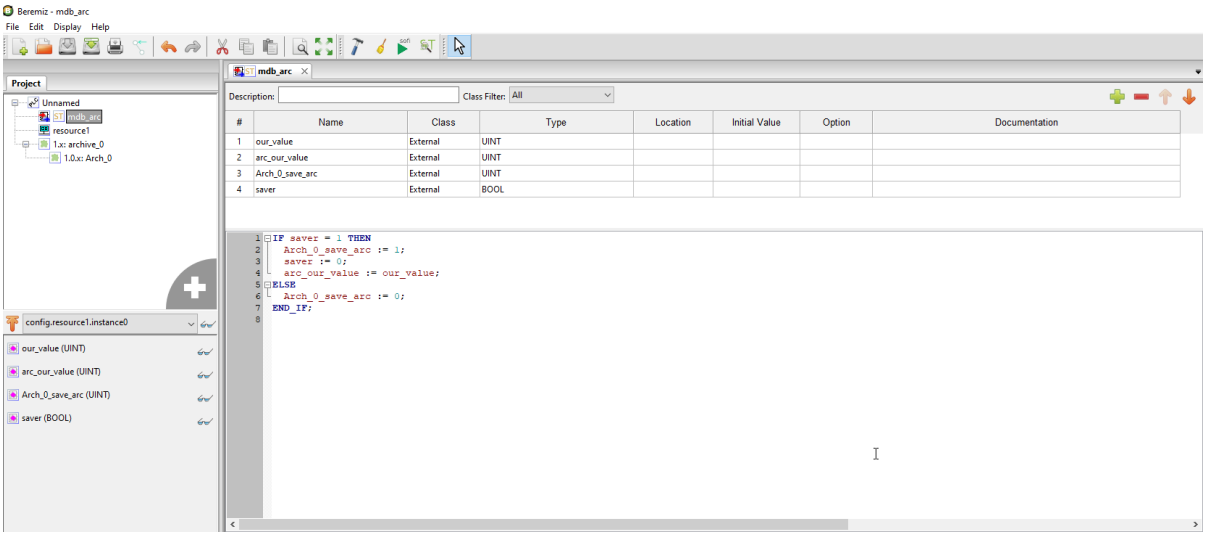


Рис. 24: *Программа mdb_arc*

Для того, чтобы через WEB-страницу менять данные переменной *our_value*, определим как *External*. Переменную *arc_our_value* добавим в окно регистров архива. Обратите внимание на то, что Modbus-адрес прописывается автоматически (50027).

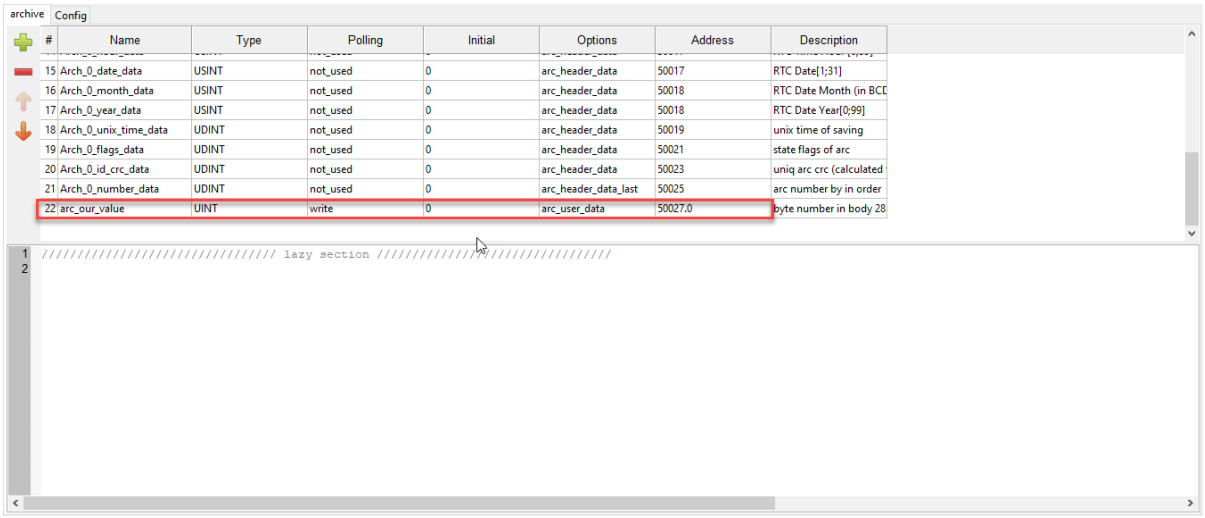


Рис. 25: *Окно регистров архива*

Переменные *saver* и *our_value* добавим в окно глобальных переменных. После компиляции программы, загружаем ее в контроллер.

Подключаемся к контроллеру BRIC через WEB-страницу, а также через порт RS-485. В нашем случае через Modbus-порт мы подключимся с помощью преобразователя интерфейсов USB-HART/RS-485. Также запустим приложение Modbus Poll с настройками, представленными ниже.

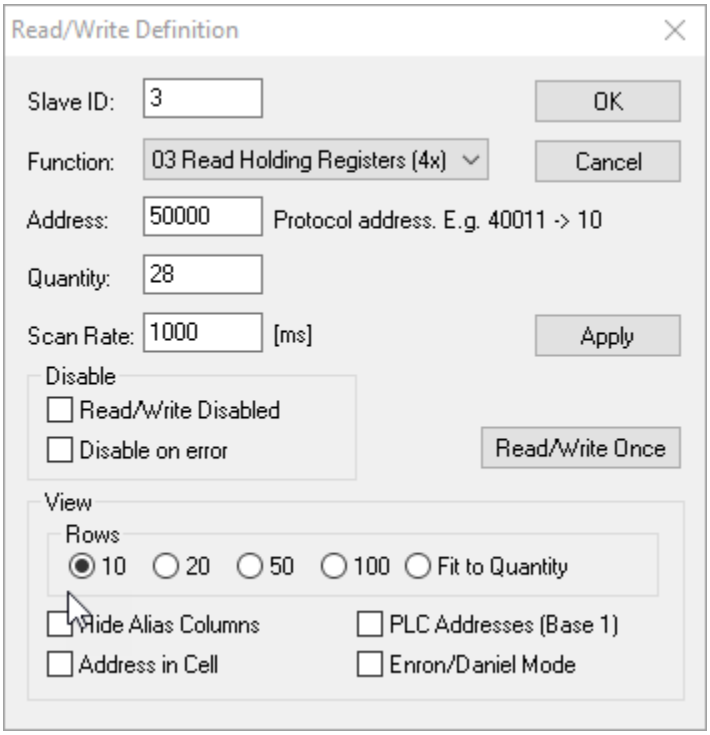
Подключение к преобразователю осуществляется как показано на рисунке ниже.

Примечание: Джамперы преобразователя должны быть настроены на RS-485:

- RS-485-RES - OFF
- +24V - OFF
- INTERFACE - RS-485

Реализация программы представлена ниже.

При записывании данных в переменную *OUR_VALUE* в Modbus-адресе 50027 нет изменений. Как только мы сохраняем в архив с помощью переменной *SAVER* - данные переменной *OUR_VALUE* появляются в Modbus Poll. Также можно заметить остальные значения Archieve Header.



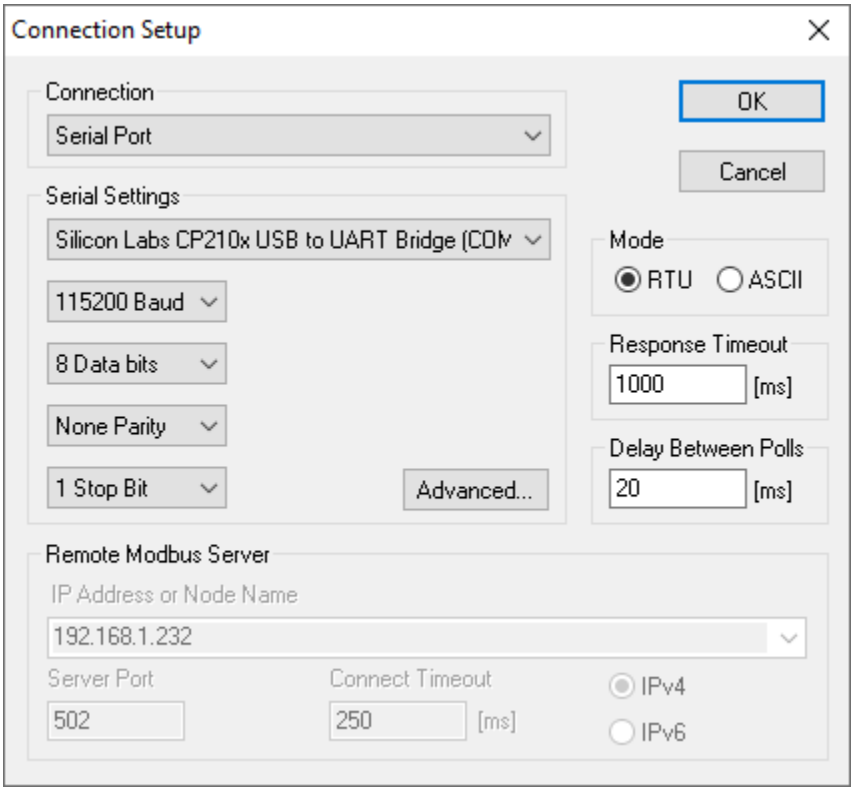


Рис. 26: *Настройка соединения в Modbus Poll*

Рис. 27: *Подключение преобразователя*

Рис. 28: *Реализация программы*

5.6 Тестовая программа. Несанкционированный доступ

Напишем простую программу контроля доступа. В данном уроке мы используем модуль архива, но подход к нему будет немного другим. Основную программу напишем на языке ST для более удобного включения записей в архив.

Итак, для начала добавим модуль архива *1.x: archive_0* и в нем подмодуль *1.0.x: Arch_0*. Основную программу назовем *test_enter*. Создаем переменную *Door_alarm*, которая будет представлять собой включение/выключение системы контроля доступа. Добавим переменные *Door1*, *Door2*, *Door3*, *Door4*. Состояние переменной *Door_status* будет результатом логического «ИЛИ» всех дверей.

Далее создаём функциональный блок на языке ST и назовем его *archivator*.

Примечание: Для создания функционального блока необходимо левой клавишой мыши нажать в облась проекта и выбрать из списка «Function block». Название блока пишется в «POU Name», в «Language» выбирается язык программирования. Для добавления функционального блока в программу необходимо зажать левой клавишей мыши в область наименования и перетащить в редактируемую область программы, задать название для него.

В данный функциональный блок добавим регистр *Arch_0_save_arc* с типом данных *UINT*. Также туда добавим уже созданные переменные *Door1*, *Door2*, *Door3*, *Door4*. Для каждого статуса дверей создаем переменную для записи в архив - *Door1_arc*, *Door2_arc*, *Door3_arc*, *Door4_arc*.

Осталось решить с какой цикличностью будет происходить запись в архив. В нашем случае выберем цикл в 2 секунды, то есть если система контроля доступа включена, начинается запись в архив данных о статусе дверей с периодичностью в 2 секунды. Для этого в программе *test_enter* добавим функциональный блок STRUCT_REAL_TIME и выберем оттуда только SEC_TIME. Даем наименование функциональному блоку - *real_time* и добавим переменную *seconds* в формате USINT. Далее с помощью функционального блока MOD вычисляем кратность переменной *seconds* к 2 и переписываем его. Функциональный блок *archivator* добавим в основную программу и даем название *arch*. Реализация программы представлена на рисунках ниже.

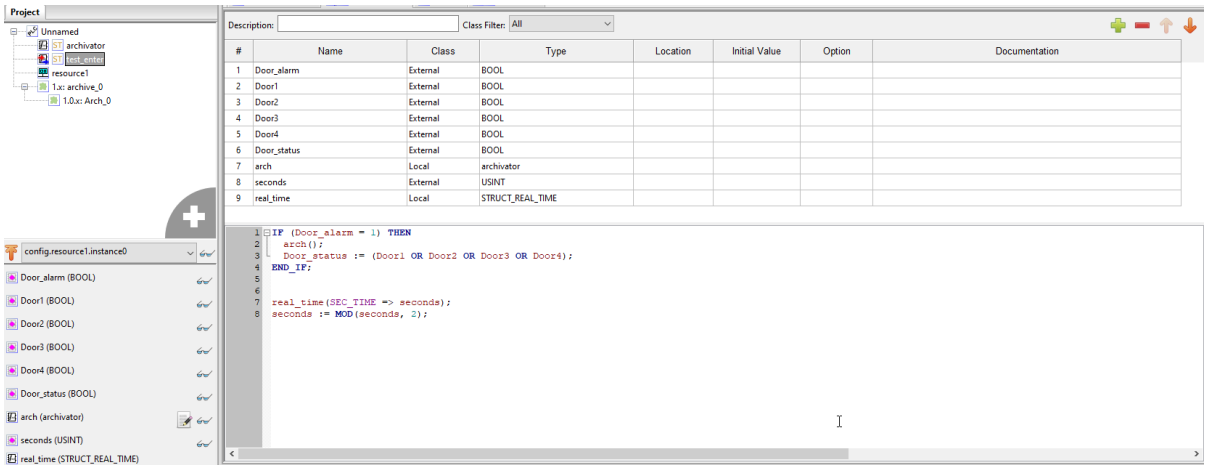


Рис. 29: *Основная программа test_enter*

Проверка программы: включаем систему контроля с помощью *DOOR_ALARM*, стартует запись в

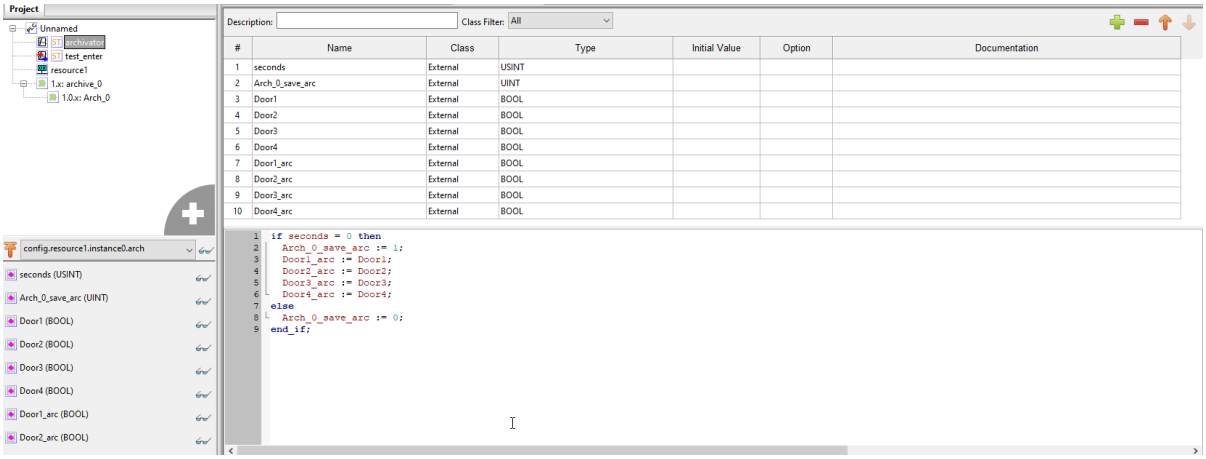


Рис. 30: *Функциональный блок archivator*

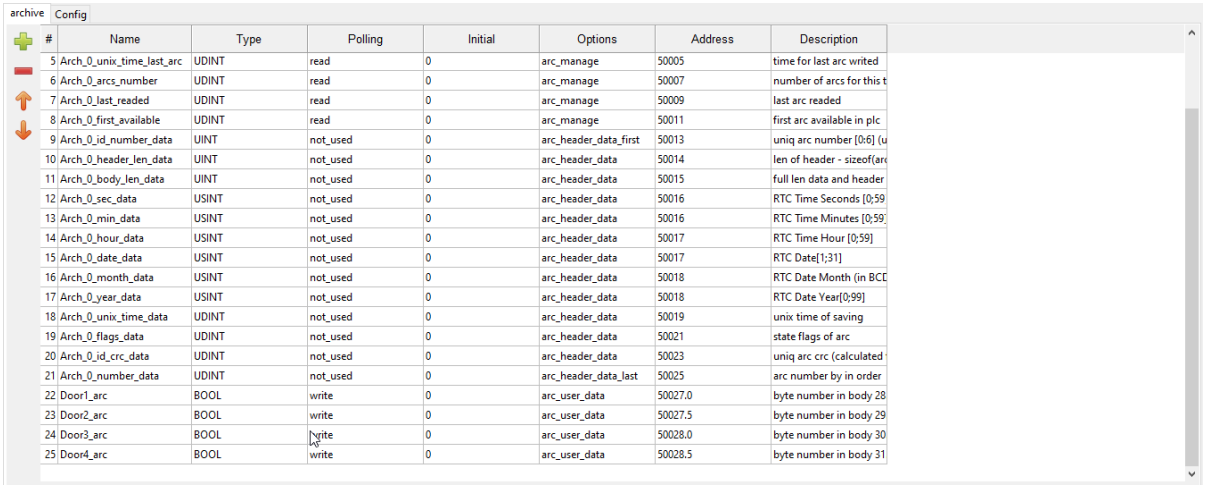


Рис. 31: *Окно регистров архива*

архив с цикличностью 2 секунды. «Открываем» и «закрываем» любые двери, информация о статусе дверей записывается в архив.

Рис. 32: *Реализация программы*

Результат открытия/закрытия дверей по времени можно увидеть на графике архивов.

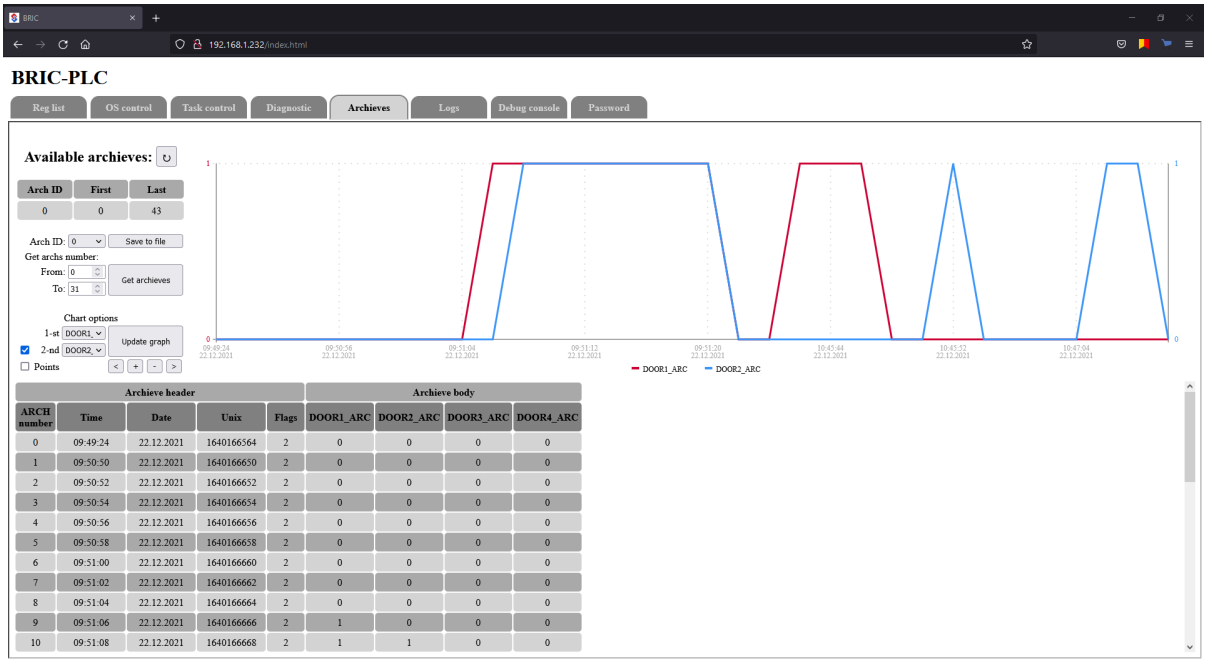


Рис. 33: *Графики архива*

5.7 Тестовая программа. Мониторинг температуры

Для данного урока необходим датчик температуры. В нашем случае будем использовать датчик температуры ТПУ 0304/М2-Н фирмы ЭЛЕМЕР с заводской установкой НСХ Pt100 dt:(-50...150)°С с внешним источником питания, канал AI_1 - пассивный.

Рис. 34: *Подключение датчика температуры ТПУ 0304/М2-Н фирмы ЭЛЕМЕР к ПЛК*

Аналогично уроку по подключению датчика температурного канала (*AI. Подключение датчика температуры 4-20 мА. Преобразование в инженерные единицы с масштабированием*) напишем программу на языке FBD, а также программу архива на языке ST.

Аналогично подключим датчик температуры на аналоговый вход AI_1. В основной программе добавим функциональный блок STRUCT_REAL_TIME. Определим время цикла записи в архив как 1 минута. Добавим переменные *m_time* и *temperature*. Финальная версия основной программы *test_temp* представлена на рисунке ниже.

Также напишем простую программу *archive* на языке ST, с цикличной записью данных температруы в 1 минуту (m_time). Данные будут записываться в переменную *temperature_arc*.

Компилируем и загружаем прошивку в ПЛК и открываем вкладку «Archieves». Данные температуры можно увидеть в графике.

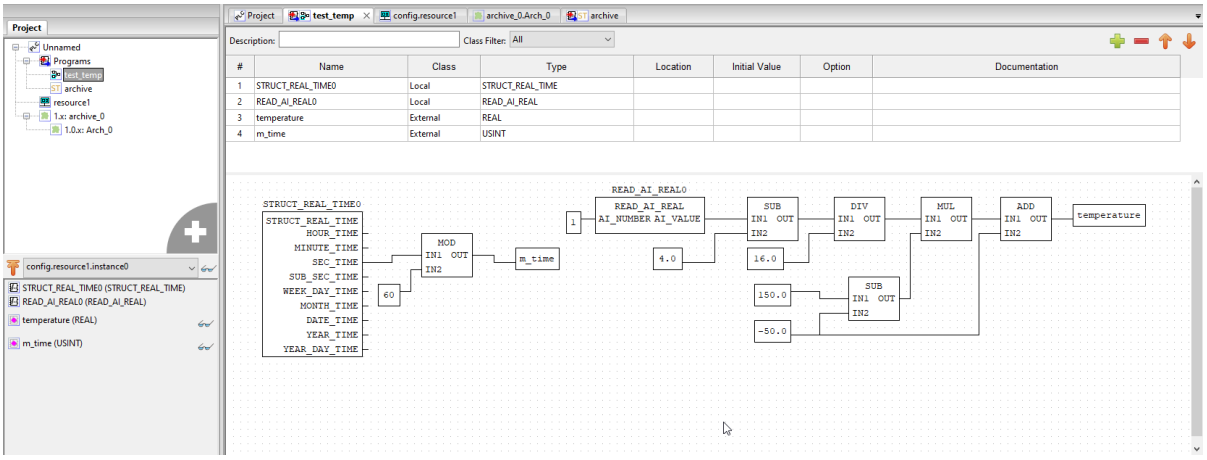


Рис. 35: *Основная программа test_ temp*

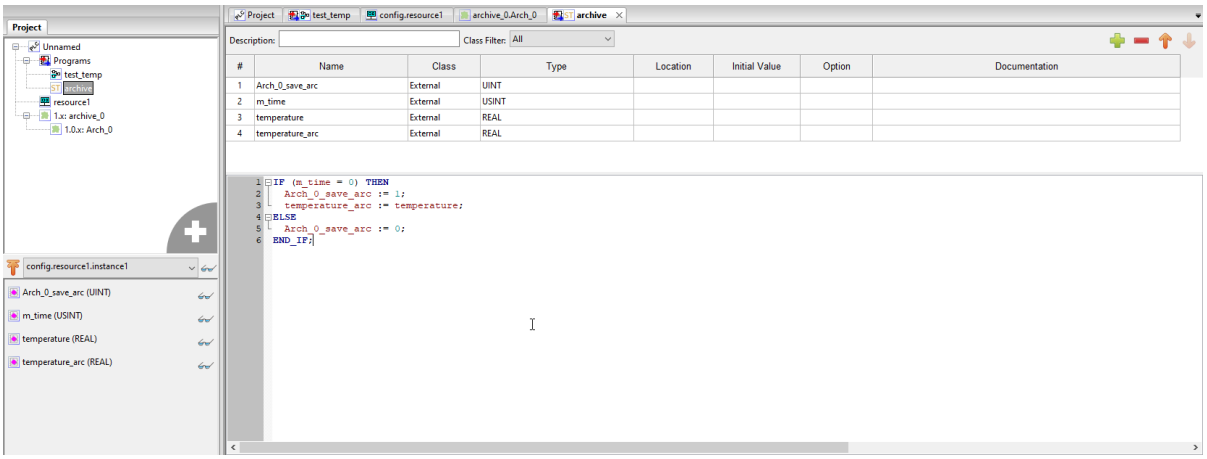


Рис. 36: *Программа archive*

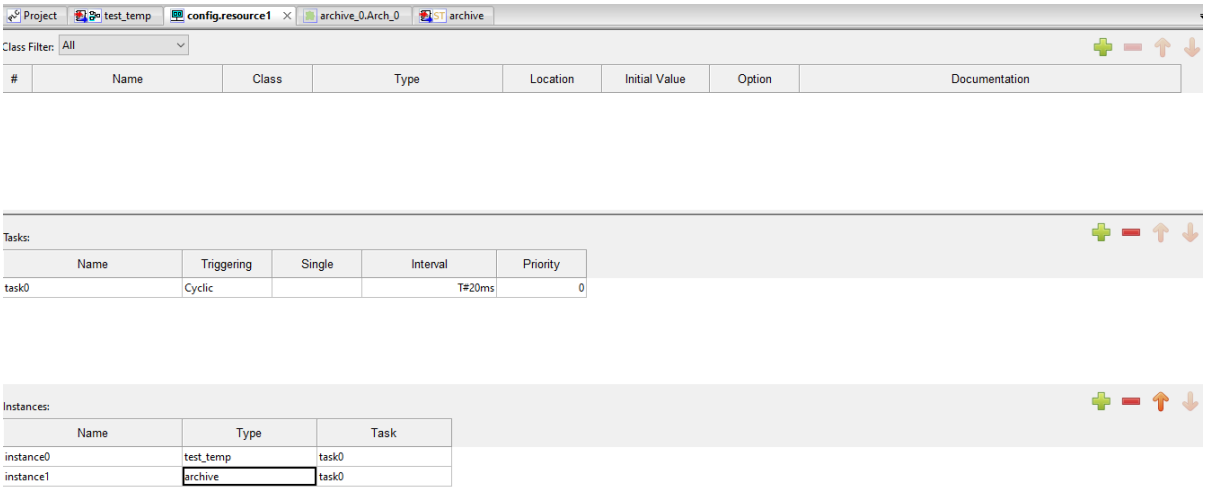


Рис. 37: *Ресурсы программ*

Archieve header					Archieve body
ARCH number	Time	Date	Unix	Flags	TEMPERATURE_ARC
0	14:39:00	22.12.2021	1640183940	2	24.482
1	14:40:00	22.12.2021	1640184000	2	24.430
2	14:41:00	22.12.2021	1640184060	2	26.927
3	14:42:00	22.12.2021	1640184120	2	31.002
4	14:43:00	22.12.2021	1640184180	2	32.875
5	14:44:00	22.12.2021	1640184240	2	32.667
6	14:45:00	22.12.2021	1640184300	2	31.661
7	14:46:00	22.12.2021	1640184360	2	30.638
8	14:47:00	22.12.2021	1640184420	2	29.806
9	14:48:00	22.12.2021	1640184480	2	29.147

Рис. 38: *График изменения температуры*