СЕРЕБРУМ

YART Studio

Среда программирования контроллеров

Язык программирования C-YART

Санкт-Петербург 2018

V1.1

www.serebrum.ru

Оглавление

C-YART 3
Типы данных 3
Комментарии к коду 3
Структура кода 3
Переменные, константы и массивы 4
Математические операторы 5
Операторы сравнения 5
Оператор IF 6
Оператор SWITCH..CASE 7
Оператор FOR 7
Системные функции 7
Функции для float 8
Сдвиги Int 8
Программирование функций 9
Ссылка на видео демонстрирующие создание блока и функции: 9
Пример программирования протокола для счетчиков МЕРКУРИЙ 230 10
Описание библиотеки Merc230 10
Пример использования 13
Описание структуры 16
Исходный код блока Merc230sd 17
Функции 23
Программирование своего протокола с использованием библиотеки для электросчетчика МЕРКУРИЙ 230 25
КОНТАКТЫ ООО ТД СЕРЕБРУМ 26

C-YART

Скриптовый язык C-YART очень простой и удобный для программирования. Он предоставляет пользователю большие возможности для создания собственных функциональных блоков и функций. Все блоки библиотеки СЕРЕБРУМ написаны на этом языке.

Типы данных

В C-YART 2 базовых типа данных: Int и Float. Любой из этих типов можно использовать как Boolean, правило преобразования одно: все, что не равно 0 (или 0.0) приравнивается к значению True («Включен»), в противном случае (если равно 0 или 0.0) - приравнивается к значению False («Отключен»).

Под любую созданную переменную в C-YART всегда выделяется 4 байта памяти (32 бита). Поэтому очень важно понимать, может ли вход или выход блока, объявленный как Byte или Short, выйти за границы допустимых значений. Чтобы избежать проблем выхода значения за допустимый предел, настоятельно рекомендуем при обучении использовать только базовые типы данных (Int и Float).

Комментарии к коду

Комментарии значительно улучшают читаемость Вашего кода. Комментарии не влияют на объем кода программы. В C-YART текст, находящийся справа от // и до конца строки считается комментарием. Фрагмент текста одной строчки или нескольких строк, заключенный между символами /* Текст комментария */, также является комментарием.

Структура кода

Весь код состоит из строк и блоков. Идентификатором окончания строки кода является знак точки с запятой « ; ». Идентификаторами начала и окончания блока являются открывающаяся и закрывающаяся фигурные скобки « { » и « } ».

Допускается использование переноса строк в любом исполнении без разрыва имен операторов, функций, переменных и других ключевых слов. Ниже представлены примеры равнозначных записей.

Пример 1:
```
if (a == b) {
    b = 0;
    c = a;
}
```

Пример 2:
```
if (a == b) { b = 0; c = a; }
```

Пример 3:
```
if (a == b)
{
    b = 0;
    c = a;
}
```

Пример 4:
```
{
    if (a == b) {
        b = 0;
        c = a;
    }
}
```

Переменные, константы и массивы

Переменные — это именованные ячейки памяти, в которые записывается результат вычислений или из которых считывается значение.

В C-YART для объявления переменной используется оператор new:

Объявление переменных типа Int:
```
new i;                  // Объявили переменную i
new y = 11;             // Объявили переменную y и присвоили ей значение 11 (тип Int)
const ABCDEF = 5;       // Объявили константу с типом Int
```

Отдельного внимания требуют только переменные с плавающей точкой - тип Float:

Объявление переменных типа Float:
```
new SetPointTemp1 = 137.1;      // Транслятор автоматически присвоит тип этой переменной Float
new Float:SetPointTemp2;        // Явно задали тип Float
const Float:ABC = 0.001;        // Объявили константу с типом Float
```

Для приведения к типу Float используется функция Float():

Приведение Int к Float:
```
new i = 11;
new Float: y;
y = Float(i) + 12.3;
```

Чтобы привести переменную типа Float к целочисленному типу используйте функцию округления floatround():

Приведение Float к Int:
```
new i ;
new y = 11.37;
i = floatround(y, floatround_round);
```

Второй аргумент функции - метод округления. Он может принимать следующие значения:

floatround_round - стандартное математическое округление (если дробная часть меньше 0,5 - аргумент округляется в меньшую сторону).

floatround_floor - округление до ближайшего целого числа в меньшую сторону (-0,001 => -1)

floatround_ceil - округление до ближайшего целого числа в большую сторону (0,001 => 1)

floatround_tozero - отбрасывается дробная часть аргумента

Переменные одного типа могут быть объединены в массив. Обращение к элементам массива производится по индексу, указанному в квадратных скобках:

```
Работа с массивами:
new y[4];              // Объявляем пустой массив y из 4-х элементов
y[2] = 888;            // Присваиваем третьему элементу значение
new r[] = [1,4,6,2,4,6];   // Присваиваем значения при объявлении массива
```

Обратите внимание, нумерация элементов массива начинается с нуля - 0,1,2,3.... Таким образом, для массива с количеством элементов 5, самый старший индекс - 4!

Математические операторы

* умножение

/ деление

- вычитание

-- декремент

+ сложение

++ инкремент

= присвоение

Операторы сравнения

Оператор сравнения возвращают результат типа bool. Например, 15 > 10 вернет - да (true).

< меньше

> больше

<= меньше или равно

>= больше или равно

== равно

!= не равно

Оператор IF

Условный оператор проверяет условие и выполняет ветвление программы - выполнение нужной части кода.

Синтаксис оператора IF:

```
If (условие) {
        // Этот код выполняется, если условие выполняется
} else {
        // Это код выполняется, если условие не выполняется
}
```

Пример использования оператора IF:

```
new x = 12;
new Float: y;
if (x > 13) {       // Условие ложно
    y = 11.3;
} else {
    y = 9;          // Будет выполнен этот код
}
```

Так как условие (x > 13) ложно, y примет значение 9.

Условие может состоять из нескольких условий, связанных между собой логическими операторами:

&& - логическое И

|| - логическое ИЛИ

Синтаксис оператора IF с несколькими условиями:

```
If (((условие1) && (условие2)) || (условие3)) {
        // Этот код выполняется, если одновременно выполняется и условие1 и условие2
        // или выполняется условие3 (условие1 и/или условие2 могут не выполнятся)
} else {
        // В противном случае выполняется этот код
}
```

Оператор SWITCH..CASE

Оператор switch позволяет в зависимости от значения аргумента выбрать блок выполняемого кода программы.

Синтаксис оператора SWITCH..CASE:

```
switch(переменная) {
    case значение1, значение2:
        // Код программы
    case значение3: {
        // Код программы
        // Код программы
    }
default:
    // Код программы, если значение не перечислено
}
```

Оператор FOR

Цикл for выполняет код, заключенный в цикле, заданное число раз. Циклы удобно применять для перебора массивов.

Синтаксис оператора FOR:

```
for (new счетчик; условие завершения; действие)
{
    // Повторяющийся код (тело цикла)
}
```

Пример использования оператора FOR:

```
new y[5];
for (new i =0; i < sizeof(y); i++)
{
    y[i] = i;        // Присваиваем каждому элементу массива текущее значение i
}
```

Объявляем цикл со счетчиком переменной `i`. Условие выхода из цикла - когда `i` равно размеру массива (т.е. `i == sizeof(y)` не попадет в программу цикла). `i++` увеличивает `i` на каждом цикле на 1. Для счетчика можно написать любое выражение: `i = i + 15;` После выполнения цикла из примера выше, получим массив `y[0,1,2,3,4]`.

ВНИМАНИЕ!!! При работе с массивами контролируйте, чтобы индекс обращения к элементу массива был меньше размера этого массива.

Для преждевременного выхода из цикла используйте оператор `return`.

Системные функции

`sizeof(x)` - размер массива, объявленного внутри блока, или являющегося внутренним параметром

`sizeof_x` - размер массива, являющегося входным или выходным параметром

GetScanDt - системная переменная, содержащая продолжительность выполнения предыдущего цикла контроллера в миллисекундах

Функции для float

floatabs(x) - модуль числа

floatfract(x) - выделить дробную часть

floatround(x, floatround_round) - округление

floatround_round - стандартное математическое округление (если дробная часть меньше 0,5 - аргумент округляется в меньшую сторону).

floatround_floor - округление до ближайшего целого числа в меньшую сторону (-0,001 => -1)

floatround_ceil - округление до ближайшего целого числа в большую сторону (0,001 => 1)

floatround_tozero - отбрасывается дробная часть аргумента

floatpower(x, pow) - возведение x в степень pow

floatsqroot(x) - корень квадратный из x, x должен быть больше нуля!

floatsin(x, radian) - синус

floatcos(x, radian) - косинус

floattan(x, radian) - тангенс

floatasin(x) - арксинус

floatacos(x) - арккосинус

floatatan(x) - арктангенс

floatlog(x, 10.0) - десятичный логарифм, x должен быть больше нуля!

ВНИМАНИЕ!!! Для корректной работы с типом float операции типа Y += 37; должны быть записаны как Y +=37.0;

Сдвиги Int

>> - арифметический сдвиг вправо e1 на e2 бит с сохранением знака (y = e1 >> e2)

<< - арифметический сдвиг влево e1 на e2 бит с сохранением знака (y = e1 << e2)

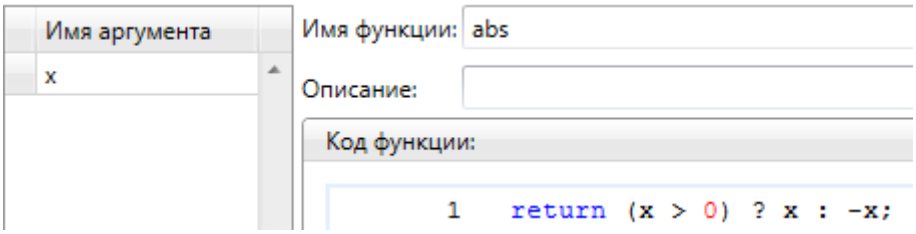
>>> - Логический сдвиг вправо e1 на e2 бит без сохранения знака (y = e1 >>> e2);

Программирование функций

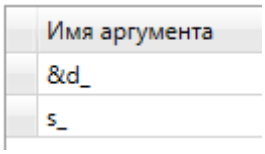
Для оптимизации используемой памяти и упрощения читаемости кода можно выносить часто используемые участки кода в функции.

Для этого необходимо в разделе «Библиотека проекта» нажать кнопку «Добавить функцию». Задайте имя функции, добавьте входящие аргументы и напишите исполняемый код функции.

Код функции должен возвращать значение. Для этого воспользуйтесь либо ключевым словом return,

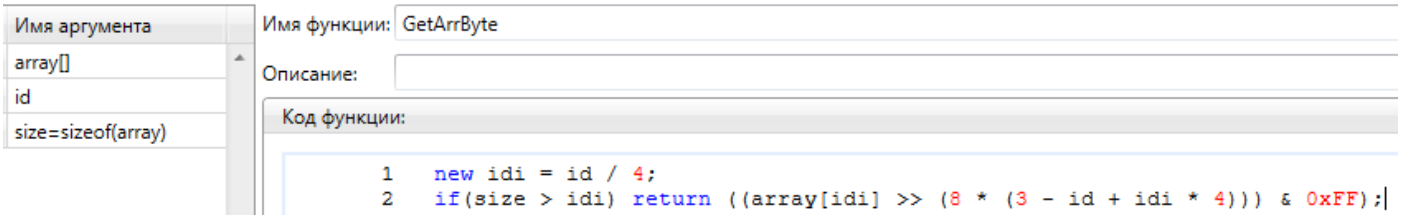


либо укажите префиксом аргумента знак &:



В этом случае можно не использовать слово return - изменение значения аргумента внутри функции приведет к изменению его значения за пределами функции (объявлена ссылка на переменную).

Для передачи в функцию массива необходимо после имени аргумента добавить квадратные скобки [] (пробела не должно быть между именем аргумента и скобками, а также между самими скобками). Если в коде функции необходимо использовать значение размера массива, то его необходимо передать входящим аргументом:



Обратите внимание, что массив является ссылочным типом. Поэтому если вы измените значение элемента массива внутри функции - оно изменится и за пределами этой функции (условно можно полагать, что перед именем массива системой автоматически подставляется знак & - взять адрес переменной (язык С)).

Вызов функции внутри блока (на примере функции abs(x) из первого рисунка):

```
new x = -5;          // Создали переменную с отрицательным значением
new y = abs(x);      // Создали переменную со значением по модулю переменной x
x = abs(x);          // Сделали значение переменной x положительным
```

Ссылка на видео демонстрирующие создание блока и функции:
https://youtu.be/oO9RuPonV9s?list=PLkWUcAjR_lovnB2-UMLlT5OzlrtTjXvks

Пример программирования протокола для счетчиков МЕРКУРИЙ 230

Описание библиотеки Merc230

Merc230 содержит два функциональных блока: Merc230sd и Merc230i и один блок настройки параметров соединения SerialPortParam.

Функциональный блок Merc230sd собирает значения статических и динамических параметров счетчика:

- NA – сетевой адрес счетчика;
- SN – серийный номер счетчика (представлен массивом из 4 байт);
- MDy – дата производства счетчика (год);
- MDm – дата производства счетчика (месяц);
- MDd – дата производства счетчика (день);
- SV – версия ПО счетчика (представлена массивом из 3 байт);
- A – постоянная счетчика;
- KTu – коэффициент трансформации счетчика по напряжению;
- KTi – коэффициент трансформации счетчика по току;
- DCy – текущая дата счетчика (год);
- DCm – текущая дата счетчика (месяц);
- DCd – текущая дата счетчика (день);
- DCwd – текущая дата счетчика (день недели: 1–7);
- TCh – текущее время счетчика (часы);
- TCm – текущее время счетчика (минуты);
- TCs – текущее время счетчика (секунды);
- F – частота электрической сети, Гц;
- UA – напряжение на 1–й фазе, В;
- UB – напряжение на 2–й фазе, В;
- UC – напряжение на 3–й фазе, В;
- IA – сила тока на 1–й фазе, А;
- IB – сила тока на 2–й фазе, А;
- IC – сила тока на 3–й фазе, А;
- PT – мощность активная по сумме фаз;
- PA – мощность активная по 1–й фазе;
- PB – мощность активная по 2–й фазе;
- PC – мощность активная по 3–й фазе;
- QT – мощность реактивная по сумме фаз;
- QA – мощность реактивная по 1–й фазе;
- QB – мощность реактивная по 2–й фазе;
- QC – мощность реактивная по 3–й фазе;
- ST – мощность комплексная по сумме фаз;
- SA – мощность комплексная по 1–й фазе;

- SB – мощность комплексная по 2–й фазе;
- SC – мощность комплексная по 3–й фазе;
- CosFiT – коэффициент мощности по сумме фаз;
- CosFiA – коэффициент мощности по 1–й фазе;
- CosFiB – коэффициент мощности по 2–й фазе;
- CosFiC – коэффициент мощности по 3–й фазе;

Кроме перечисленных выходов у блока имеется еще два логических выхода типа Boolean:

- «netError» – переходит во включенное состояние при отсутствии устойчивой связи со счетчиком;
- «dateError» – переходит во включенное состояние при несоответствии текущей даты счетчика и контроллера.

Для работы блоку требуются следующие входные параметры:

- inEnable – включение/ выключение блока (Boolean);
- inPortPar – параметры коммуникации, задаются в конфигурационном блоке (Short);
- inMercAddress – сетевой адрес опрашиваемого счетчика (Byte);
- inPassword – пароль для подключения в режиме 01 для опрашиваемого счетчика (Byte[6]);
- inTimeSync – включение/выключение автоматической коррекции времени (Boolean);
- inOutOfSyncSeconds – разность времени между счетчиком и контроллером (в секундах), при превышении которой должна проводиться автоматическая коррекция времени (Byte);
- bufIn – буфер для входящих сообщений (Link);
- bufOut – буфер для исходящих сообщений (Link);

Функциональный блок Merc230i собирает накопленные значения:

- EAp – Энергия активная, прямая;
- EAn – Энергия активная, обратная;
- ERp – Энергия реактивная, прямая;
- ERn – Энергия реактивная, обратная;
- EAT1p – Энергия активная, прямая, по тарифу 1;
- EAT1n – Энергия активная, обратная, по тарифу 1;
- ERT1p – Энергия реактивная, прямая, по тарифу 1;
- ERT1n – Энергия реактивная, обратная, по тарифу 1;
- EAT2p – Энергия активная, прямая, по тарифу 2;
- EAT2n – Энергия активная, обратная, по тарифу 2;
- ERT2p – Энергия реактивная, прямая, по тарифу 2;
- ERT2n – Энергия реактивная, обратная, по тарифу 2;
- EAT3p – Энергия активная, прямая, по тарифу 3;
- EAT3n – Энергия активная, обратная, по тарифу 3;
- ERT3p – Энергия реактивная, прямая, по тарифу 3;
- ERT3n – Энергия реактивная, обратная, по тарифу 3;
- EAT4p – Энергия активная, прямая, по тарифу 4;

- EAT4n – Энергия активная, обратная, по тарифу 4;
- ERT4p – Энергия реактивная, прямая, по тарифу 4;
- ERT4n – Энергия реактивная, обратная, по тарифу 4;
- ELAp – Энергия технических потерь активная, прямая;
- ELAn – Энергия технических потерь активная, обратная;
- ELRp – Энергия технических потерь реактивная, прямая;
- ELRn – Энергия технических потерь реактивная, обратная;

Для работы блоку требуются следующие входные параметры:

- inEnable – включение/ выключение блока (Boolean);
- inPortPar – параметры коммуникации, задаются в конфигурационном блоке (Short);
- inMercAddress – сетевой адрес опрашиваемого счетчика (Byte);
- inPassword – пароль для подключения в режиме 01 для опрашиваемого счетчика (Byte[6]);
- bufIn – буфер для входящих сообщений (Link);
- bufOut – буфер для исходящих сообщений (Link);

Более подробная информация о входящих параметрах представлена в таблице:

Параметр	Тип	Минимальное значение	Максимальное значение	Значение по умолчанию
inEnable	Boolean	false	true	false
inPortPar	Short	нет	нет	нет
inPassword	Byte[6]	[1, 1, 1, 1, 1, 1]	[255,255,255,255,255,255]	[1,1,1,1,1,1]
inTimeSync	Boolean	false	true	False
inOutOfSyncSeconds	Byte	1	220	10

Блок настройки параметров соединения SerialPortParam используется для формирования значения типа Short, содержащего в себе всю информацию, требующуюся для связи контроллера и счетчика через интерфейс RS485 (порт контроллера, количество бит данных, параметры четности, количество стоповых бит и скорость обмена данными).

Для работы блоку требуются следующие входные параметры:

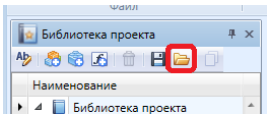
- Port – идентификатор используемого порта (Byte);
- DataBits – количество бит данных (Byte);
- Parity – используемый тип четности (Byte);
- StopBits – количество стоповых бит (Byte);
- Baudrate – скорость работы порта (Byte);

Параметры задаются не значениями, а индексами. Более детальная информация о формате значения на выходе и соответствия входных индексов реальным значениям представлена в таблице:

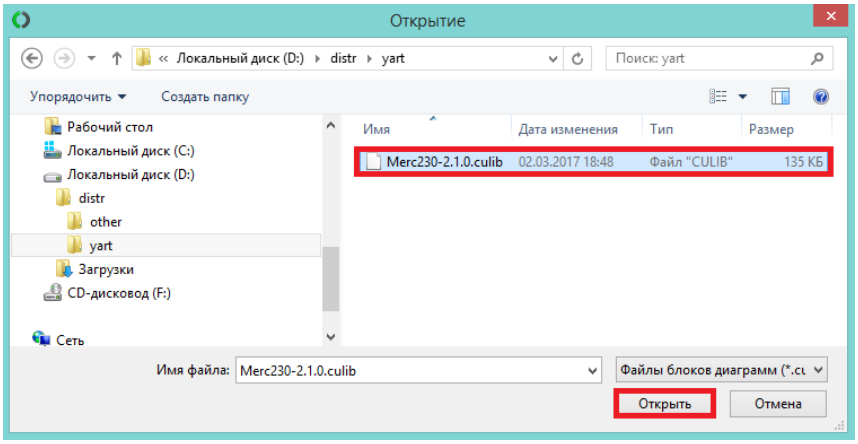
Baudrate				StopBits		Parity			DataBits			Port			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	err			0	1	0	N		0	8		0	err		
1	1200			1	2	1	O		1	7		1	1		
2	2400			2	1,5	2	E		2	6		2	2		
3	4800					3	M		3	5		3	3		
4	9600					4	S		4	err		4	резерв		
5	19200					5	err		5	err		5	резерв		
6	38400					6	err		6	err		6	резерв		
7	57600					7	err		7	err		7	резерв		
8	115200											8	резерв		
9	резерв											9	резерв		
10	резерв											10	резерв		
11	резерв											11	резерв		
12	резерв											12	резерв		
13	резерв											13	резерв		
14	резерв											14	резерв		
15	резерв											15	резерв		

Пример использования

Перед использованием библиотеки необходимо её импортировать (в случае если блоков нет в основной библиотеке) – откройте раздел «Библиотека проекта» и нажмите кнопку «Загрузить элемент»:

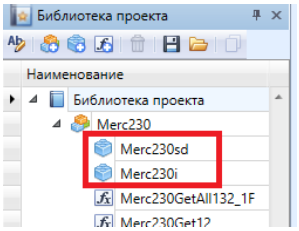


Откроется диалоговое окно выбора библиотеки:

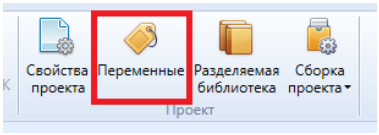


Выберите библиотеку Merc230 и нажмите кнопку «Открыть». В результате в разделе «Библиотека проекта» появится папка «Merc230», в которой будут находиться два функциональных

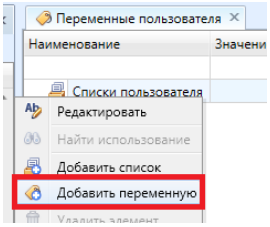
блока (Merc230sd и Merc230i), один блок настройки параметров соединения (ComSettings) и вспомогательные функции:



Для работы функциональных блоков необходимо создать вспомогательные переменные (для входящего и исходящего буферов). Для этого войдите в раздел «Переменные пользователя», нажав кнопку основного интерфейса «Переменные»:



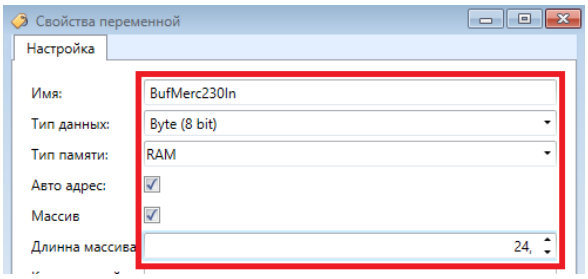
В появившемся разделе нажмите правой кнопкой мыши на строку «Списки пользователя» и в появившемся меню выберите пункт «Добавить переменную»:



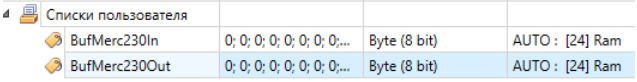
В окне «Свойства переменной» задайте значения для полей:

- «Имя» – любое уникальное имя переменной, идентифицирующее ее принадлежность к входящему буферу модуля Merc230;
- «Тип данных» – Byte;
- «Тип памяти» – RAM;
- «Авто адрес» – рекомендуем оставить отмеченным;
- «Массив» – установите отметку;
- «Длинна массива» – 24.

Подтвердите создание переменной кнопкой «OK».



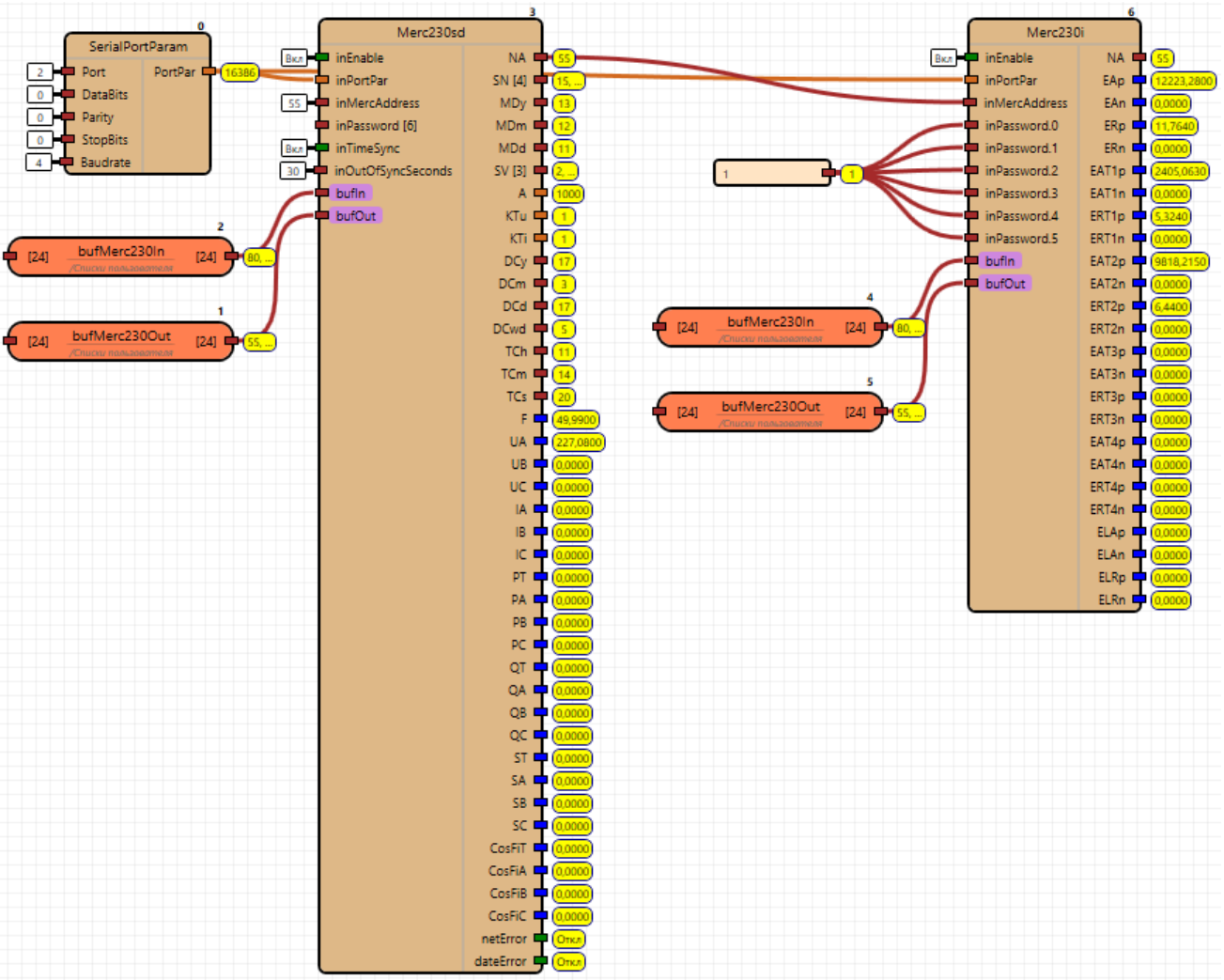
Аналогичным образом создайте переменную для исходящего буфера. В результате у Вас должно быть объявлено две переменные:



Далее необходимо в окне программы разместить необходимое количество функциональных блоков, опрашивающих счетчики. Для примера опросим 2 счетчика и считаем с них значения всех параметров (будем использовать оба функциональных блока).

Добавим на поле один блок настройки параметров соединения SerialPortParam, блок Merc230sd и блок Merc230i. Поле «PortPar» конфигурационного блока соединим с полем «inPortPar» функциональных блоков. Добавим на поле созданные ранее переменные для буфера и соединим их с входами «bufIn» и «bufOut» наших блоков. Поле «inEnable» зададим константой для всех блоков. Поле «inMercAddress» будем задавать константой только у блока Merc230sd, для блока Merc230i подадим на вход – выход «NA» блока Merc230sd, хотя можно задать константой и его.

Для демонстрации возможности использования не стандартного пароля – на крайнем правом блоке мы подали на вход полей «inPassword» значение созданной константы. Если необходима автоматическая синхронизация времени счетчика с контроллером – устанавливаем «inTimeSync» константой во включено и по желанию указываем максимально допустимую рассинхронизацию в секундах в поле «inOutOfSyncSeconds», после которой будет инициирован процесс коррекции времени:



Описание структуры

Библиотека Merc230 содержит два функциональных блока: Merc230sd и Merc230i и один блок настройки параметров соединения SerialPortParam. Кроме того, в состав библиотеки Merc230 входит ряд функций, обеспечивающих работу функциональных блоков, эти функции вызываются из функциональных блоков.

Разделение библиотеки на два функциональных блока обусловлено ограничением компилятора YART Studio, не позволяющим добавлять в блок неограниченное число переменных входов/выходов - общая длинна имен перечисленных, учитывая разделяющие символы (запятая и пробел), не должно превышать 950 символов.

В блоке Merc230sd - реализован более сложный алгоритм отслеживания ошибок. В блоке Merc230i механизм отслеживания ошибок упрощен.

Рассмотрим блок Merc230sd. Входные переменные блока:

inEnable	Bool	inTimeSync	Bool
inPortPar	Short (16...	inOutOfSyncSeconds	Byte (8 bit)
inMercAddress	Byte (8 bit)	bufIn	Byte (8 bit)
inPassword [6]	Byte (8 bit)	bufOut	Byte (8 bit)

Выходные переменные блока:

NA	Byte (8 bit)	F	Float (32 bit)	ST	Float (32 bit)
SN [4]	Byte (8 bit)	UA	Float (32 bit)	SA	Float (32 bit)
MDy	Byte (8 bit)	UB	Float (32 bit)	SB	Float (32 bit)
MDm	Byte (8 bit)	UC	Float (32 bit)	SC	Float (32 bit)
MDd	Byte (8 bit)	IA	Float (32 bit)	CosFiT	Float (32 bit)
SV [3]	Byte (8 bit)	IB	Float (32 bit)	CosFiA	Float (32 bit)
A	Short (16...	IC	Float (32 bit)	CosFiB	Float (32 bit)
KTu	Short (16...	PT	Float (32 bit)	CosFiC	Float (32 bit)
KTi	Short (16...	PA	Float (32 bit)	netError	Bool
DCy	Byte (8 bit)	PB	Float (32 bit)	dateError	Bool
DCm	Byte (8 bit)	PC	Float (32 bit)		
DCd	Byte (8 bit)	QT	Float (32 bit)		
DCwd	Byte (8 bit)	QA	Float (32 bit)		
TCh	Byte (8 bit)	QB	Float (32 bit)		
TCm	Byte (8 bit)	QC	Float (32 bit)		
TCs	Byte (8 bit)				

Внутренние переменные:

_init	Bool	_MDd	Byte (8 bit)	_DCy	Byte (8 bit)
_blockId	Short (16...	_SV [3]	Byte (8 bit)	_DCm	Byte (8 bit)
_stage	Byte (8 bit)	_KTu	Short (16...	_DCd	Byte (8 bit)
_readTime	Float (32 bit)	_KTi	Short (16...	_DCwd	Byte (8 bit)
_msgId	Byte (8 bit)	_F	Int (32 bit)	_TCh	Byte (8 bit)
_skipMsgCount	Byte (8 bit)	_U [4]	Int (32 bit)	_TCm	Byte (8 bit)
_netErr	Byte (8 bit)	_I [4]	Int (32 bit)	_TCs	Byte (8 bit)
_A	Short (16...	_P [4]	Int (32 bit)	_timeSync	Bool
_SN [4]	Byte (8 bit)	_Q [4]	Int (32 bit)	_timeDif	Int (32 bit)
_MDy	Byte (8 bit)	_S [4]	Int (32 bit)	_netError	Bool
_MDm	Byte (8 bit)	_CosFi [4]	Int (32 bit)	_dateError	Bool

Исходный код блока Merc230sd

В первую очередь нам необходимо прочитать параметры из конфигурационного блока:

```
new inPortId, inFlags, inBaudrate;
Merc230GetComSets(inPortPar, inPortId, inFlags, inBaudrate);
```

Далее блок выполнит инициализацию (при первом запуске алгоритма контроллера), в которой внутренней переменной `_blockId` присваивается значение порядкового номера функционального блока, опрашивающего заданный последовательный порт (для каждого порта создается собственный список порядковых номеров функциональных блоков). В этом сегменте кода можно разместить любой пользовательский код, который должен исполняться только один раз – при первом обращении к функциональному блоку (он будет исполнен даже если входящий параметр `inEnable == false`):

```
if(!_init){
    _blockId = getCountPortId(inPortId);
    setCountPortId(inPortId, _blockId + 1);
    _init = true;
}
```

После инициализации объявляем условные константы:

```
new defOutOfSyncSeconds = 10;       // секунд
new readTimeout         = 2000;     // миллисекунд
new timeGetMsgId        = 17;       // ид сообщения получения времени
new timeSyncMsgId       = 18;       // ид сообщения синхронизации времени
new ya_YY = Merc230ShortLToInt(90); // дата контроллера - год
new ya_MM = Merc230ShortRToInt(88); // дата контроллера – месяц
new ya_DD = Merc230ShortLToInt(88); // дата контроллера - день
new ya_WD = Merc230ShortRToInt(90); // дата контроллера – день недели
new ya_hh = Merc230ShortLToInt(86); // время контроллера - часы
new ya_mm = Merc230ShortRToInt(86); // время контроллера - минуты
new ya_ss = Merc230ShortLToInt(84); // время контроллера – секунды
// проставляем пароль по умолчанию (все символы единицы), если он не был
// указан (если в пароле все символы нули)
new sumPasByte = 0;
for(new i = 0; i < 6; i++) sumPasByte +=inPassword[i];
if(sumPasByte == 0) for(new i = 0; i < 6; i++) inPassword[i] = 1;
```

Объявляем переменные:

```
new crc;
```

Считываем параметры ссылочных переменных (буферы):

```
new bufInAdr = bufIn[0];
new bufInLen = bufIn[1];
```

```
new bufOutAdr = bufOut[0];
new bufOutLen = bufOut[1];
```

Следующий фрагмент кода определяет потерю устойчивой связи с электросчетчиком и очищает значения внутренних переменных, если более чем 2 раза подряд от счетчика не был получен корректный ответ. Очистка значений делается для того, чтобы при потере связи – функциональный блок прекращал отображение последних полученных значений. Все параметры для очистки перечислять не нужно, а только динамические. После потери связи функциональный блок продолжит публиковать его сетевой адрес, серийный номер и т.д.:

```
if(_netErr > 2){
    netError = true;
    _F = 0.0;
    for(new i = 0; i < 4; i++){
        _P[i] = 0.0;
        _Q[i] = 0.0;
        _S[i] = 0.0;
    }
    for(new i = 0; i < 3; i++){
        _U[i] = 0.0;
        _I[i] = 0.0;
    }
}else{
netError = false;
}
```

Отобразим на выходе блока действующие значения переменных (чтение и запись массивов выполняется через циклы). У некоторых параметров есть коэффициенты, корректирующие значения. Для параметров с плавающей точкой к целому коэффициенту необходимо дописывать .0):

```
NA = inMercAddress;
for(new i = 0; i < 4; i++) SN[i] = _SN[i];
MDy = _MDy;
MDm = _MDm;
MDd = _MDd;
for(new i = 0; i < 3; i++) SV[i] = _SV[i];
A = _A;
KTu = _KTu;
KTi = _KTi;
DCy = _DCy;
DCm = _DCm;
DCd = _DCd;
DCwd = _DCwd;
TCh = _TCh;
TCm = _TCm;
TCs = _TCs;
F = _F / 100.0;
UA = _U[0] / 100.0;
UB = _U[1] / 100.0;
UC = _U[2] / 100.0;
IA = _I[0] / 1000.0;
IB = _I[1] / 1000.0;
IC = _I[2] / 1000.0;
PT = _P[0] / 100.0;
PA = _P[1] / 100.0;
PB = _P[2] / 100.0;
PC = _P[3] / 100.0;
QT = _Q[0] / 100.0;
QA = _Q[1] / 100.0;
QB = _Q[2] / 100.0;
QC = _Q[3] / 100.0;
ST = _S[0] / 100.0;
```

```
SA = _S[1] / 100.0;
SB = _S[2] / 100.0;
SC = _S[3] / 100.0;
CosFiT = _CosFi[0] / 1000.0;
CosFiA = _CosFi[1] / 1000.0;
CosFiB = _CosFi[2] / 1000.0;
CosFiC = _CosFi[3] / 1000.0;
dateError = _dateError;
```

После отображения всех действующих значений параметров мы можем приступить к проверке на необходимость дальнейшей работы блока (все блоки опрашиваются по очереди и нужно проверять в том ли блоке мы находимся, который должен опрашиваться в данный момент):

```
if(getCurrentPortId(inPortId) != _blockId) return;
```

Если именно этот блок должен выполняться, но входящий параметр `inEnable` не равен `true`, то нужно перейти к опросу следующего блока:

```
if(getCurrentPortId(inPortId) == _blockId && !inEnable){
    _msgId = 0;
    NextPortId(inPortId);
    return;
}
```

Далее начинается основная работа блока – опрос устройства. В первую очередь формируем массив исходящих сообщений:

```
new msgOut[][] =    [
    [0x00],                         // 0// тест канала связи [1-2]
    [0x01,0x01,0x00,0x00,0x00,0x00,0x00,0x00],// 1// установка соединения [8-2]
    [0x08,0x12],                    // 2// вариант исполнения [2-7]
    [0x08,0x00],                    // 3// получение серийного номера [2-8]
    [0x08,0x03],                    // 4// версия ПО счетчика [2-4]
    [0x08,0x02],                    // 5// коэффициенты трансформации [2-5]
    [0x08,0x11,0x40],               // 6// частота сети [3-4]
    [0x08,0x11,0x11],               // 7// напряжение на фазе 1[3-4]
    [0x08,0x11,0x12],               // 8// напряжение на фазе 2[3-4]
    [0x08,0x11,0x13],               // 9// напряжение на фазе 3[3-4]
    [0x08,0x11,0x21],               //10// ток на фазе 1[3-4]
    [0x08,0x11,0x22],               //11// ток на фазе 2[3-4]
    [0x08,0x11,0x23],               //12// ток на фазе 3[3-4]
    [0x08,0x16,0x00],               //13// мощность P [3-13]
    [0x08,0x16,0x04],               //14// мощность Q [3-13]
    [0x08,0x16,0x08],               //15// мощность S [3-13]
    [0x08,0x16,0x30],               //16// коэффициент мощности CosFi [3-13]
    [0x04,0x00],                    //17// внутреннее время счетчика [2-9]
    [0x03,0x0D,0x00,0x00,0x00]      //18// коррекция времени [5-2]
    ];
```

Создаем два массива с длинами запросов и ответов (длинна запроса не учитывает сетевой адрес устройства и CRC, длинна ответа не учитывает CRC):

```
new msgOutLength[] = [1,8,2,2,2,2,3,3,3,3,3,3, 3, 3, 3, 3,2,5];
new msgInLength[]  = [2,2,7,8,4,5,4,4,4,4,4,4,13,13,13,13,9,2];
```

Добавим код, который переходит к следующему запросу, если превышено время ожидания текущего. Если на третий запрос электросчетчик не дал корректного ответа функциональный блок останавливает опрос:

```
_readTime += GetScanTaskEx;
if(_readTime > readTimeout){
    _skipMsgCount ++;
    _netErr ++;
    if(_msgId == 0 || _skipMsgCount > 2){
        _skipMsgCount = 0;
        _msgId = sizeof(msgOut) - 1;
    }
```

```
    _stage = 5;
}
```

Код переключателя отслеживающего текущую стадию (структура switch/case):

```
switch(_stage){

    …

}
```

В первую очередь необходимо открыть порт для опроса устройства, после чего сбросить внутренний буфер входящих сообщений:

```
case 0:{
    port_open(inPortId, inBaudrate, inFlags);
    stop_port_receive(inPortId);
    _stage = 1;
}
```

Подготавливаем сообщение к отправке и отправляем его электросчетчику:

```
case 1:{
    // для второго сообщения (первое имеет индекс 0) заполняем массив паролем
    if(_msgId == 1) for(new i = 0; i < 6; i++) msgOut[_msgId][i + 2] =
inPassword[i];
    // служебный флаг, требуется для корректной синхронизации времени
    if(_msgId == timeGetMsgId){
        _timeSync = false;
    }
    // подсчет времени смещения при синхронизации
    if(_msgId == timeSyncMsgId){
        new mm_ = 0;
        new hh_ = 0;
        new tD = (_timeDif < 0) ? (_timeDif + 60) / 60 : _timeDif / 60;
        new ss = _TCs + _timeDif - tD * 60;
        new mm = _TCm + tD;
        new hh = _TCh;
        if(ss >= 60) mm_ = ss / 60; else if(ss < 0) mm_ = (ss + 60) / 60 - 1;
        if(mm >= 60) hh_ = mm / 60; else if(mm < 0) hh_ = (mm + 60) / 60 - 1;
        if(ss >= 60) ss -= (ss / 60) * 60; else if(ss < 0) ss += 60;
        if(mm >= 60) mm -= (mm / 60) * 60; else if(mm < 0) mm += 60 - ((mm +
60) / 60) * 60;
        mm += mm_;
        hh += hh_;

        msgOut[_msgId][2] = Merc230IntToHex(ss);
        msgOut[_msgId][3] = Merc230IntToHex(mm);
        msgOut[_msgId][4] = Merc230IntToHex(hh);
        _timeSync = false;
    }

    // Запись текущего сообщения в буфер
    set_byte(bufOutAdr, inMercAddress);
    for(new i = 0; i < msgOutLength[_msgId]; i++){
        set_byte(bufOutAdr + 1 + i, msgOut[_msgId][i]);
    }

    // Расчет CRC
    crc = Merc230GetCRC(bufOutAdr,msgOutLength[_msgId] + 1);

    set_byte((bufOutAdr + msgOutLength[_msgId] + 1), (crc & 0xFF));
    set_byte((bufOutAdr + msgOutLength[_msgId] + 2), (crc >> 8));

    // Отправка сообщения
    if(port_transmit_status(inPortId) == 0){
        stop_port_receive(inPortId);
        port_receive(inPortId, bufInAdr, bufInLen);
```

```
            port_send(inPortId, bufOutAdr, msgOutLength[_msgId] + 3);
            _stage = 2;
        }
    }
```

Ожидаем окончания отправки и получаем ответ от устройства:

```
    // Ожидание окончания отправки сообщения
    case 2:{
        if(port_transmit_status(inPortId) == 0){
            _stage = 3;
        }
    }

    // Ожидание окончания получения ответа
    case 3:{
        if(port_receive_status(inPortId) == (bufOutLen - msgInLength[_msgId] - 2)){
            stop_port_receive(inPortId);
            _stage = 4;
        }
    }
```

Обработаем ответ от устройства. Выполним подсчет CRC. Если CRC правильный, то сбрасывается счетчик пропущенных сообщений и происходит вход в переключатель формирования выходных данных. Если CRC неправильный у первого сообщения (тест канала связи) – мы моментально уходим из данного блока (флаг ошибки связи в данном случае установиться после третьей ошибки CRC подряд):

```
    case 4:{

        // Расчет CRC
        crc = Merc230GetCRC(bufInAdr,msgInLength[_msgId]);

        // Проверка CRC
        if((get_byte(bufInAdr + msgInLength[_msgId]) == crc & 0xFF) &&
            (get_byte(bufInAdr + msgInLength[_msgId] + 1) == crc >> 8)){
            _skipMsgCount = 0;

            // Формирование выходных данных
            switch(_msgId){

                …   // Дальнейший код нужно вставлять сюда

            }
        }else if(_msgId == 0){
            _netErr ++;
            _msgId = sizeof(msgOut) - 1;
        }
        _stage = 5;
    }
```

В сегменте формирования выходных данных анализируем полученный ответ, прошедший проверку по CRC. Переключатель основан на ID сообщения и прописывает алгоритм действия для каждого сообщения (где необходим анализ ответа). Первым анализируем ответ теста канала связи, если в начале ответа указан сетевой адрес устройства, и следующим байтом идет 0x00, значит тест пройден и можно сбросить флаг ошибки связи, если он был выставлен ранее, если же в ответе ошибка – уходим из блока:

```
case 0:
    if(get_byte(bufInAdr) != inMercAddress || get_byte(bufInAdr + 1) != 0x00){
        _netErr ++;
        NextPortId(inPortId);
        return;
    } else _netErr = 0;
```

Далее анализируются ответы на последующие сообщения и считываются значения параметров из входящего буфера (если требуется однотипная обработка ответа – лучше вывести ее в функцию):

```
case 2:{
    switch(get_byte(bufInAdr + 2) & 0x0F){
        case 0: _A = 5000;
        case 1: _A = 25000;
        case 2: _A = 1250;
        case 3: _A = 500;
        case 4: _A = 1000;
        case 5: _A = 250;
    }
}
case 3: {
    for(new i = 0; i < 4; i++) _SN[i] = get_byte(bufInAdr + i + 1);
    _MDy = Merc230HexToInt(get_byte(bufInAdr + 7));
    _MDm = Merc230HexToInt(get_byte(bufInAdr + 6));
    _MDd = Merc230HexToInt(get_byte(bufInAdr + 5));
}
case 4: for(new i = 0; i < 3; i++) _SV[i] = Merc230HexToInt(get_byte(bufInAdr +
i + 1));
case 5: {
    _KTu = Merc230Get12(bufInAdr,0);
    _KTi = Merc230Get12(bufInAdr,2);
}
case 6: _F = Merc230Get132(bufInAdr,0);
case 7: _U[0] = Merc230Get132(bufInAdr,0);
case 8: _U[1] = Merc230Get132(bufInAdr,0);
case 9: _U[2] = Merc230Get132(bufInAdr,0);
case 10: _I[0] = Merc230Get132(bufInAdr,0);
case 11: _I[1] = Merc230Get132(bufInAdr,0);
case 12: _I[2] = Merc230Get132(bufInAdr,0);
case 13: Merc230GetAll132_1F(_P,bufInAdr,4);
case 14: Merc230GetAll132_1F(_Q,bufInAdr,4);
case 15: Merc230GetAll132_1F(_S,bufInAdr,4);
case 16: Merc230GetAll132_1F(_CosFi,bufInAdr,4);
case 17: {
    _DCy = Merc230HexToInt(get_byte(bufInAdr + 7));
    _DCm = Merc230HexToInt(get_byte(bufInAdr + 6));
    _DCd = Merc230HexToInt(get_byte(bufInAdr + 5));
    _DCwd = Merc230HexToInt(get_byte(bufInAdr + 4));
    _TCh = Merc230HexToInt(get_byte(bufInAdr + 3));
    _TCm = Merc230HexToInt(get_byte(bufInAdr + 2));
    _TCs = Merc230HexToInt(get_byte(bufInAdr + 1));
    if((_DCy > 0 || _DCm > 0 || _DCd > 0) &&
        _TCh > 1 && _TCh < 23 &&
```

```
            (ya_YY != _DCy || ya_MM != _DCm || ya_DD != _DCd)) _dateError = true;
        else _dateError = false;

        // расчет разницы во времени для синхронизации времени
        if(inOutOfSyncSeconds < 1 || inOutOfSyncSeconds > 230) inOutOfSyncSeconds =
    defOutOfSyncSeconds;
        _timeDif = (ya_hh - _TCh) * 3600 + (ya_mm - _TCm) * 60 + ya_ss - _TCs;
        if(inTimeSync && ya_YY == _DCy && ya_MM == _DCm && ya_DD == _DCd &&
            abs(_timeDif) >= inOutOfSyncSeconds){
                _timeSync = true;
                if(_timeDif < -220) _timeDif = -220;
                else if(_timeDif > 220) _timeDif = 220;
        }
    }
```

Программа для формирования выходных данных готова – далее необходимо в корневой переключатель (`switch(_stage)`) добавить заключительный этап, в котором мы переходим к отправке следующего сообщения (если не выставлен флаг `_timeSync`, то сообщение с `_msgId == timeSyncMsgId` пропускается), сбрасываем счетчик времени, и переходим к следующему блоку, если завершена обработка последнего сообщения:

```
    case 5:{
        _msgId++;
        _readTime = 0;
        if(_msgId == timeSyncMsgId  && !_timeSync) _msgId++;
        if(_msgId == sizeof(msgOut)){
            _msgId = 0;
            _stage = 0;
            NextPortId(inPortId);
        }else{
            _stage = 1;
        }
    }
```

Функции

Merc230Get132(`bufAdr, offset`) – считывает из буфера (`bufAdr`) значение в виде целого числа, читая побайтно в порядке 1-3-2 байты со смещением (`offset`) для разбора длинных строк:

```
    return  (get_byte(bufAdr + 1 + offset * 3) << 16) |
            (get_byte(bufAdr + 3 + offset * 3) << 8) |
             get_byte(bufAdr + 2 + offset * 3);
```

Merc230Get132_1F(`bufAdr, offset`) – считывает из буфера (`bufAdr`) значение в виде целого числа, читая побайтно в порядке 1-3-2 байты с применением битового И 0x1F к первому байту со смещением (`offset`) для разбора длинных строк:

```
    return  ((get_byte(bufAdr + 1 + offset * 3) & 0x1F) << 16) |
            (get_byte(bufAdr + 3 + offset * 3) << 8) |
             get_byte(bufAdr + 2 + offset * 3);
```

Merc230GetAll132_1F(`var[], bufAdr, count`) – считывает `count` элементов в массив значений (`var[]`) в порядке 1-3-2 с применением битового И 0x1F к первому байту:

```
    for(new i = 0; i < count; i++) var[i] = Merc230Get132_1F(bufAdr, i);
```

Merc230Get2143(`bufAdr, offset`) – считывает из буфера (`bufAdr`) значение в виде целого числа, читая побайтно в порядке 2-1-4-3 байты со смещением (`offset`) для разбора длинных строк:

```
    return  (get_byte(bufAdr + 2 + offset * 4) << 24) |
            (get_byte(bufAdr + 1 + offset * 4) << 16) |
            (get_byte(bufAdr + 4 + offset * 4) << 8) |
```

```
            get_byte(bufAdr + 3 + offset * 4);
```

Merc230GetAll2143(var[], bufAdr, count) – считывает count элементов в массив значений (var[]) в порядке 2-1-4-3, если значение > 0:

```
for(new i = 0; i < count; i++){
    new val = Merc230Get2143(bufAdr, i);
    if(val > 0) var[i] = val;
}
```

Merc230Get12(bufAdr, offset) – считывает из буфера (bufAdr) значение в виде целого числа, читая побайтно в порядке 1-2 байты со смещением (offset) для разбора длинных строк:

```
return (get_byte(bufAdr + 1 + offset) << 8) | get_byte(bufAdr + 2 + offset);
```

Merc230GetCRC(bufAdr, msgLength) – рассчитывает CRC для строки длинной msgLength в буфере bufAdr:

```
new crcTab[256], crcT0, crcT1;
for (new i = 0; i < 256; i++){
    crcT0 = 0;
    crcT1 = i;
    for (new j = 0; j < 8; j++){
        if(((crcT0 ^ crcT1) & 0x0001) > 0) crcT0 = (crcT0 >> 1) ^ 0xA001;
        else crcT0 = crcT0 >> 1;
        crcT1 = crcT1 >> 1;
    }
    crcTab[i] = crcT0;
}

new crc = 0xFFFF;
new ptr = get_byte(bufAdr);
crc = (crc >> 8) ^ crcTab[(crc ^ (0x00FF & ptr)) & 0xFF];
for (new i = 1; i < msgLength; i++){
    ptr = get_byte(bufAdr + i);
    crc = (crc >> 8) ^ crcTab[(crc ^ ptr) & 0xFF];
}
return crc;
```

Merc230ShortLToInt(adr) – преобразует первый байт переменной типа Short, расположенной по адресу adr, в целое число так, чтобы 0x57 был равен 57:

```
return (get_short(adr) >> 12) * 10 + ((get_short(adr) >> 8) & 0x0F);
```

Merc230ShortRToInt(adr) – преобразует второй байт переменной типа Short, расположенной по адресу adr, в целое число так, чтобы 0x57 был равен 57:

```
return ((get_short(adr) >> 4) & 0x0F) * 10 + (get_short(adr) & 0x0F);
```

Merc230HexToInt(x) – по аналогии с Merc230ShortRToInt(adr) преобразует входящее число, в целое число так, чтобы 0x57 был равен 57:

```
return (x >> 4) * 10 + (x & 0x0F);
```

abs(x) – возвращает значение числа x по модулю:

```
return (x > 0) ? x : -x;
```

Merc230IntToHex(x) – функция обратная Merc230HexToInt(x) – преобразует 57 в 0x57:

```
return ((x / 10) << 4) | (x - (x / 10) * 10);
```

Merc230GetComSets(in, &port, &flags, &baudrate) – чтение параметров порта из переменной типа Short:

```
port = in & 0x0F;
new stop = (in >> 10) & 0x03;
if(stop > 1) stop = 0;
new parity = (in >> 7) & 0x07;
if(parity > 2) parity = 0;
flags = (stop << 2) | parity;
switch ((in >> 12) & 0x0F){
    case 1: baudrate = 1200;
```

```
        case 2: baudrate = 2400;
        case 3: baudrate = 4800;
        case 4: baudrate = 9600;
        case 5: baudrate = 19200;
        case 6: baudrate = 38400;
        case 7: baudrate = 57600;
        case 8: baudrate = 115200;
    }
```

Программирование своего протокола с использованием библиотеки для электросчетчика МЕРКУРИЙ 230

Изменение исходного кода начните с корректировки выходных переменных блока и создания соответствующих им внутренних переменных для хранящих значения между циклами алгоритма. Определите необходимость расчета CRC (в модуле Merc230 используется CRC MODBUS).

После этого необходимо модифицировать участки кода:

- задающие условные константы
- обнуляющие значения при потере связи и отображающие значения на выходе.

Если сообщение должно начинаться с сетевого адреса устройства, то достаточно просто скорректировать массивы сообщений (3 массива: `msgOut[][], msgOutLength[], msgInLength[]`). Если же начало сообщения не должно начинаться с адреса устройства, то необходимо удалить строку:

```
set_byte(bufOutAdr, inMercAddress);
```

И внести корректировку в следующие строки – уменьшить на единицу все случаи прибавления к `msgOutLength[]`:

```
// Запись текущего сообщения в буфер
set_byte(bufOutAdr, inMercAddress);
for(new i = 0; i < msgOutLength[_msgId]; i++){
    set_byte(bufOutAdr + i, msgOut[_msgId][i]);
}

// Расчет CRC
new crc;
crc = Merc230GetCRC(bufOutAdr,msgOutLength[_msgId]);

set_byte((bufOutAdr + msgOutLength[_msgId]), (crc & 0xFF));
set_byte((bufOutAdr + msgOutLength[_msgId] + 1), (crc >> 8));

// Отправка сообщения
if(port_transmit_status(inPortId) == 0){
    stop_port_receive(inPortId);
    port_receive(inPortId, bufInAdr, bufInLen);
    port_send(inPortId, bufOutAdr, msgOutLength[_msgId] + 2);
    _stage = 2;
}
```

Лишние строки в коде подготовки сообщения нужно удалить. После этого останется лишь исправить код формирования выходных данных в соответствии с требуемым алгоритмом сбора данных и запускать получившийся модуль на тестирование.

В случае отказа от расчета CRC или если исходящие сообщения необходимо формировать без сетевого адреса устройства – максимальное число ошибок в обмене данными может возникнуть на этапе ожидания окончания получения ответа.

КОНТАКТЫ ООО ТД СЕРЕБРУМ

ООО «ТД Серебрум»

195196 Санкт-Петербург ул. Громова д.4

Телефоны:

+7 812 976-86-86,

+7 812 715–89–13,

+7 812 970–37–58,

Факс:+7 812 648–12–80,

Сайт: www.serebrum.ru

Электронная почта: support@serebrum.ru

YouTube - https://www.youtube.com/cerebrum-automation