Тел/факс: +7 (495) 545 – 28 – 94;
Тел: +7 (495) 543 – 88 – 54;
E-mail: cybrotech@olil.ru

Руководство пользователя программы CyPro

Ред. 27

(применяется для CyPro v2.5.7 и более поздней редакции)

Olil Ltd
М.О., г.Химки, Нагорное шоссе, д.2
тел/факс: +7 495 545 28 94
тел: +7 495 543 88 54
www.cybrotech.ru
cybrotech@olil.ru

Содержание

Введение

Обзор системы

CyPro является пакетом программ для программируемых контроллеров CyBro-2. Они работают с операционной системой Microsoft Windows 2000/XP/Vista. CyPro является полностью функциональным IDE (интегрированная среда обработки) ограничивающим редактором, компилятором и оперативным монитором.

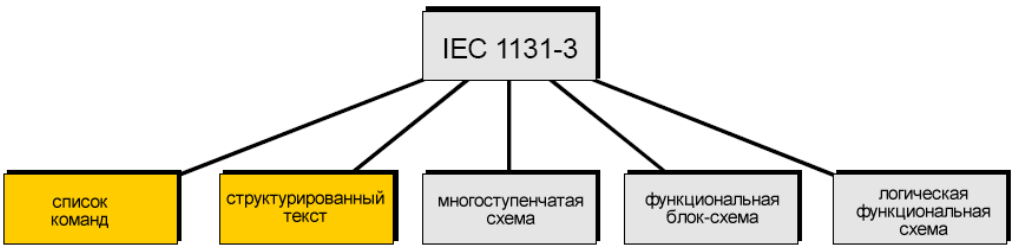
CyBro-2 является родовым именем для группы контроллеров, основанных на одних и тех же базовых технологиях. Все контроллеры совместимы, хотя некоторые особенности не могут поддерживаться.

Каждый CyBro-2 помечается уникальным 6-цифровым серийным номером, также используемым как адрес коммуникационной сети (NAD (адрес)).

CyBro-2 имеет четыре независимых коммуникационных канала:

Com1	программирование / объединение в сеть
Com2	программирование / объединение в сеть или свободно программируемый последовательный порт
ETH	программирование / объединение в сеть
IEX-2	присоединяемые платы расширения

CyPro основывается на стандарте IEC 1131-3. Текущая версия выполняет список команд и структурированную текстовую программу, расширенную с помощью нескольких визуальных инструментов.



Требования к аппаратным средствам

Достаточно иметь любой ПК, работающий с операционной системой MS Windows 2000. CyPro будет занимать на диске около 5 Мб.

Для соединения CyBro и ПК требуется стандартный последовательный порт (RS-232) или порт Ethernet. Обычно поддерживаются преобразователи USB-последовательный порт, хотя некоторые устройства могут не работать.

Установка

Для установки CyPro запускают программу установки и следуют инструкциям. Рекомендованная директория для установки C:\Program Files\CyPro. Если необходима более старая версия, рекомендуется добавить номер версии в директорию установки, например, C:\Program Files\CyPro201.

Установка выполняется следующим образом:

- В указанную директорию распаковываются файлы CyPro
- Создается группа меню запуска и значков
- Настраивается связь с файлом типа .cyp
- Настраивается язык CyPro и коммуникационный порт

Никакие файлы не копируются в директорию Windows. Никакие системные файлы Windows не заменяются и не изменяются. Директорией по умолчанию для файлов пользователя является C:\Program Files\CyPro\Project, хотя возможно открывать (Open) или сохранять (Save) проекты в любой другой директории.

Чтобы обновить CyPro установите новую версию в ту же самую директорию без удаления предыдущей версии. Настройки пользователя будут сохраняться.

При обновлении CyPro необходимо также обновить программное обеспечение (ядро) системы CyBro. Чтобы сделать это запустите Tools/Kernel Maintenance (Инструменты/Обслуживание ядра), загрузите kernel.bin и перешлите его в контроллер CyBro.

CyPro и версия ядра всегда должны быть согласованы. Новая версия CyPro всегда представляется с обновленным файлом ядра.

Для удаления CyPro запустите Control Panel (панель управления), Add/Remove Programs (добавление/удаление программ), выберите CyPro и нажмите кнопку Add/Remove (добавить/удалить).

Коммуникация

Контроллер CyBro-2 имеет два последовательных порта (RS-232) и один порт Ethernet. Все три порта физически независимы и могут работать одновременно. Все порты являются совместимыми с A-bus, хотя действительный протокол различен для последовательной и TCP/IP сети.

Для установления коммуникации между CyPro и CyBro должны быть сделаны следующие шаги:

- Откройте новый проект
- Сделайте настройки коммуникации в соответствии с конкретным соединением (последовательное или Ethernet)
- Откройте Hardware setup (установка аппаратных средств) (F5)
- Присоедините CyBro-2
- Запустите Autodetect (автоматическое распознавание) (Alt-A)

Если коммуникация не устанавливается, проверьте значок индикатора кабеля в строке состояний.

Коммуникационный порт недоступен. Причина может быть в неправильном номере порта или порт может быть готов к использованию для другого приложения.

Коммуникационный порт открыт, но кабель не определяется. Причина может быть в неправильном номере порта или коммуникационном кабеле. Присоедините кабель к другому коннектору DB-9 или используйте другой кабель.

Кабель определяется. Если коммуникация еще не работает, проверьте сетевой адрес и соединение.

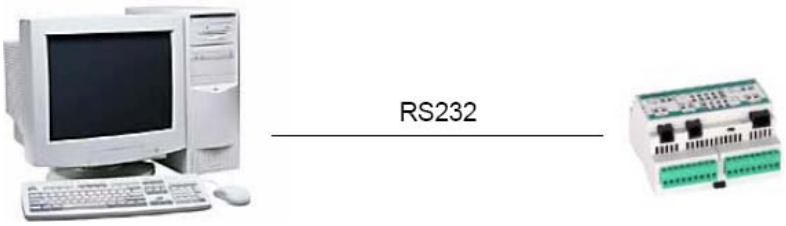
Если через последовательный интерфейс присоединяется более одного контроллера CyBro, не будет работать автоматическое распознавание NAD (адрес). В этом случае, пожалуйста, введите вручную сетевой адрес для каждого CyBro. Широковещательный адрес (нуль) никогда не используется как коммуникационный адрес.

Если активируется опция Synchronize program with PLC (программа синхронизации с контроллером), команды Start (запуск) и Monitor (монитор) будут автоматически компилироваться и посылаться текущий проект в CyBro. Эта опция размещается в Tools/Environment Options/Communication (инструменты/опции окружающей среды/коммуникация).

Для соединения ПК и CyBro имеется в распоряжении номер опций присоединения.

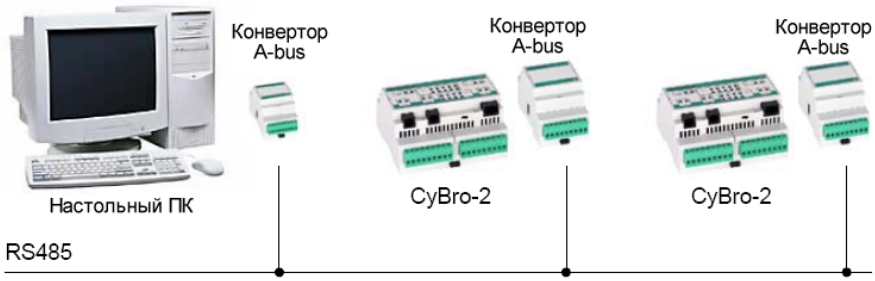
Последовательное соединение

Основное соединение использует последовательный кабель:



Соединение через USB порт с использованием преобразователя USB-последовательный порт:

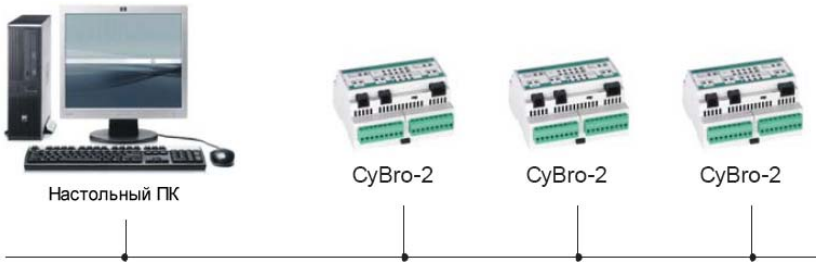
Последовательная коммуникация RS-485 с использованием преобразователей CAD-232-A2:



Соединение TCP/IP

Основное соединение с помощью кабеля Ethernet:

Присоединение по локальной сети:



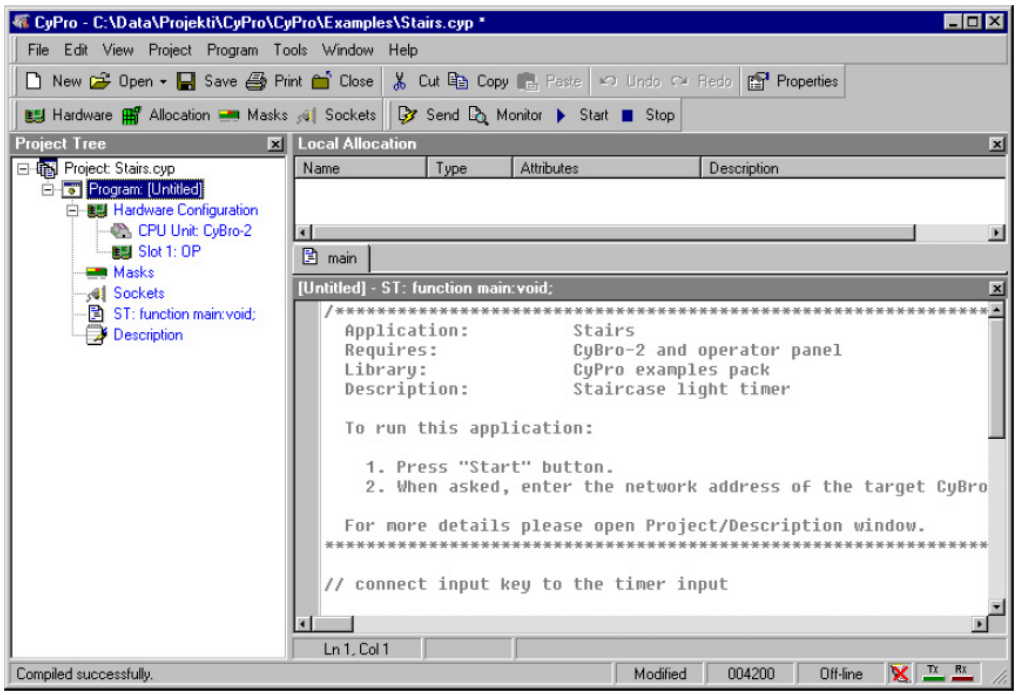
Присоединение к глобальной сети через Интернет:

Более подробно о настройке сети смотрите в разделе Объединение в сеть.

Интерфейс пользователя

Основное окно

Основное окно программы CyPro состоит из окна редактирования, панелей инструментальных средств и строки состояния. Настройка экрана по умолчанию показана ниже:



Каждая компонента может вводиться или изменяться. Для вывода перенесите компоненту левой вертикальной линией над зоной редактирования. Для обратного ввода перенесите окно в границы основного окна.

Стандартная панель инструментальных средств

New	Создает новый пустой проект
Open	Открывает существующий проект (Ctrl-O)
Save	Сохраняет текущий проект (Ctrl-S)
Print	Печатает текущий проект (Ctrl-P)
Close	Закрывает текущий проект
Cut	Удаляет выбранное и помещает его в буфер обмена (Ctrl-X)
Copy	Копирует выбранное в буфер обмена (Ctrl-C)
Paste	Устанавливает содержимое буфера обмена курсором, заменяет любое выбранное (Ctrl-V)
Undo	Отмена самого последнего действия редактирования (Ctrl-Z)

Программная панель инструментальных средств

Hardware	Открывает диалоговое окно Hardware Setup (установка аппаратных средств) (F5)
Allocation	Открывает диалоговое окно Allocation Editor (редактор выделения) (F6)
Masks	Открывает редактор Mask List (список масок) (F7)
Sockets	Открывает редактор Socket List (список гнезд) (F8)

Коммуникационная панель инструментальных средств

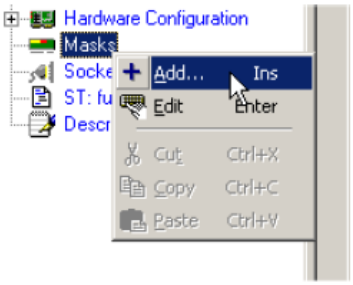
Send	Посылает текущий проект в CyBro (F9)
Monitor	Открывает оперативный Variable Monitor (монитор переменных) (F10)
Start	Запускает программу для CyBro (F11)
Stop	Останавливает программу для CyBro и отключает все выходные сигналы (F12)

Дерево проекта

Дерево проекта показывает иерархию всех частей проекта. Клик левой кнопкой на ⊞ будет раскрывать дерево и позволяет видеть дерево более детально. Для свертки дерева нажмите на ⊟.

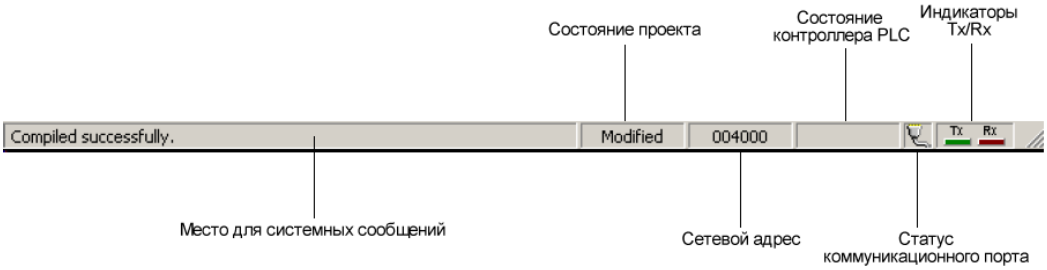
Клик правой кнопкой на любой компоненте открывает контекстное чувствительное всплывающее меню. В зависимости от типа возможно Add (добавление), Edit (редактирование), Delete (удаление) или изменение Properties (свойства) выбранной компоненты.

Например, для задания новой маски кликните правой кнопкой на Masks (маски) и выберите Add (добавить).



Строка текущего состояния

Строка текущего состояния показывает различную информацию о коммуникации и присоединенном CyBro. Строка текущего состояния находится внизу основного окна.



System messages (системные сообщения) показывают результат предшествующей операции.

Project status (состояние проекта) показывается, если текущий проект не сохраняется. Оно отражает изменения в источнике, назначение, список масок, гнезд или монитора.

Network address (сетевой адрес) показывает коммуникационный адрес выбранного в настоящее время CyBro. Кликните правой кнопкой для выбора другого адреса или введите новый адрес. Список сетевых адресов сохраняется вместе с проектом.

PLC status (состояние контроллера) показывает:

Off-line	CyBro не отвечает.
Run	CyBro подключен и работает.
Stop	CyBro подключен, остановлен. Выходы неактивны и программа не выполняется.
Pause	CyBro подключен, работа приостановлена. Выходы остаются активными, но программа не выполняется.
Error	CyBro подключен, появляется ошибка в работе. Коды ошибки указаны в приложении. Для удаления ошибки нажмите Stop (остановка).
Loader	CyBro подключен, но программное обеспечение системы (ядро) не активно. Запустите Kernel Maintenance (обслуживание ядра) и пошлите новое ядро.
Busy	ПК присоединяется к работающей сети CyBro. Для установки коммуникации, первым должна приостанавливаться работа мастера сети.

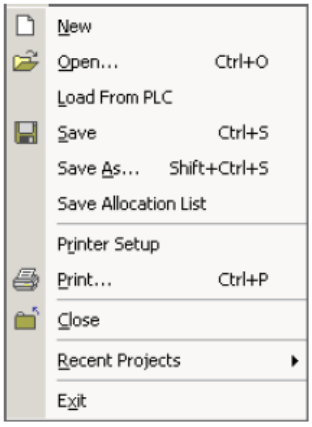
Communication port status (состояние коммуникационного порта) показывает правильность присоединения соответствующего коммуникационного кабеля:

	OK
	Коммуникационный порт в нормальном состоянии, но кабель не присоединен
	Коммуникационный порт не существует или уже используется другим приложением

Communication indicators (индикаторы коммуникации) показывают активность передачи/получения. Зеленый СИД показывает передачу (Tx) и красный СИД показывает получение (Rx).

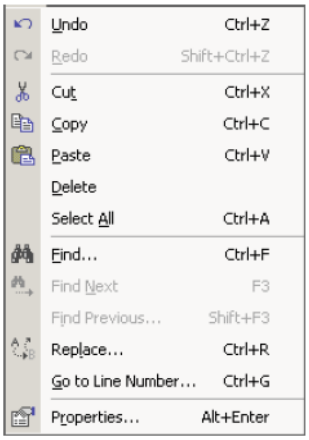
Меню

Файл



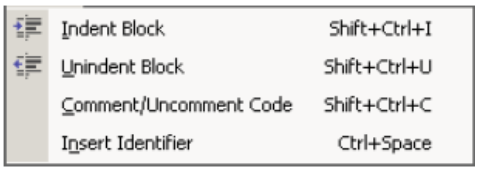
New	Создание нового проекта
Open	Открытие существующего проекта
Load From PLC	Загрузка проекта из контроллера
Save	Сохранение текущего проекта
Save As	Сохранение текущего проекта под новым именем
Save Allocation List	Сохранение списка выделений для использования с CyBro OPC или UniOP Designer
Printer Setup	Выбор принтера и опций принтера
Print	Печать текущего проекта
Close	Закрытие текущего проекта
Recent Project	Открытие недавно открытого проекта
Exit	Выход

Редактирование



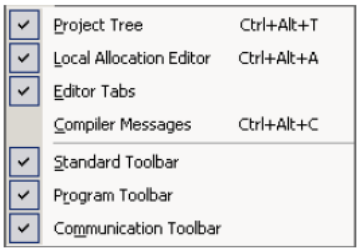
Undo	Отмена последнего действия
Redo	Отмена последней операции Undo
Cut	Удаление выбранного и помещение в буфер обмена
Copy	Копирование выбранного в буфер обмена
Paste	Вставка текста из буфера обмена в точку вставки
Delete	Удаление выбранного элемента
Select All	Выбор всего документа
Find	Нахождение специфического текста
Find Next	Нахождение следующего месторасположения специфического текста
Find Previous	Нахождение предыдущего месторасположения специфического текста
Replace	Нахождение специфического текста и замена его
Go to Line Number	Сдвиг точки вставки на указанную строку в тексте
Properties	Показ свойств выбранного модуля проекта

Формат



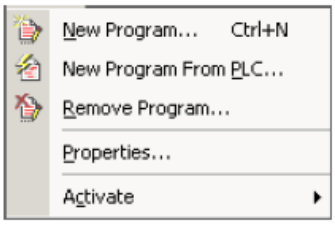
Indent Block	Блок структурирования для программы/текста
Unident Block	Блок деструктурирования для программы/текста
Comment/Uncomm. Code	Комментируемая/Некомментируемая выбранная часть программы
Insert Identifier	Открытие списка идентификаторов для вставки в программу

Просмотр



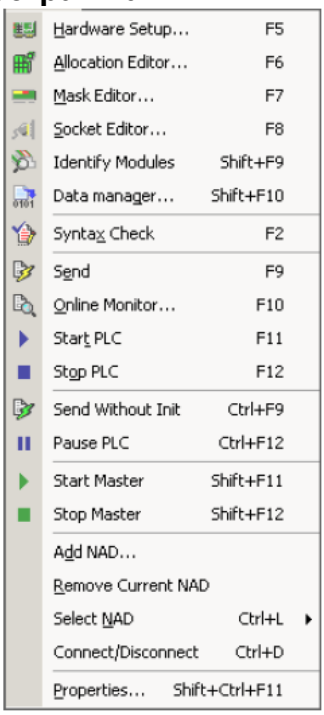
Project Tree	Показывает Project Tree (дерево проекта)
Local Allocation Editor	Показывает Local Allocation Editor (местный редактор выделения)
Editor Tabs	Показывает Editor Tabs (редакторские закладки)
Compiler Messages	Показывает Compiler Messages (сообщения о компиляции)
Standard Toolbar	Показывает Standard Toolbar (стандартная панель инструментальных средств)
Program Toolbar	Показывает Program Toolbar (программируемая панель инструментальных средств)
Communication Toolbar	Показывает Communication Toolbar (коммуникационная панель инструментальных средств)

Проект



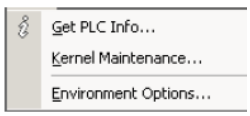
New Program	Создает New Program (новую программу) в текущем проекте
New Program From PLC	Загрузка программы из контроллера (PLC) в текущий проект
Remove Program	Удаление программы из текущего проекта
Properties	Показывает свойства текущего проекта

Программа



Hardware Setup	Открывает диалоговое окно Hardware Setup (настройка аппаратных средств)
Allocation Editor	Открывает диалоговое окно Allocation Editor (редактор расположения)
Mask Editor	Открывает редактор Mask Editor (редактор маски)
Socket Editor	Открывает редактор Socket Editor (редактор гнезда)
Identify Modules	Идентифицирует модули IEX и отдельные входы / выходы
Data Manager	Резервирование / восстановление набора переменных PLC в/из ПК
Syntax Check	Проверка текущего источника для ошибок
Send	Посылка текущей программы в CyBro
Online Monitor	Оперативное чтение/запись параметров контроллера (PLC)
Start PLC	Запуск программы CyBro
Stop PLC	Остановка программы CyBro и отключение всех выводов
Stop Without Init	Посылка текущей программы в CyBro, сохранение выходных сигналов и переменных контроллера (PLC)
Pause PLC	Пауза в программе CyBro, сохранение выходных сигналов в активном состоянии
Start Master	Запуск сетевого мастера
Stop Master	Остановка сетевого мастера
Add NAD (адрес)	Добавление нового сетевого адреса в текущую программу
Remove Current NAD	Удаление текущего NAD (адрес) из текущей программы
Select NAD (адрес)	Выбор текущего сетевого адреса для активной программы
Connect/Disconnect	Присоединение / отсоединение коммуникационного порта
Activate	Активация программы
Properties	Показ свойств программы

Инструментальные средства

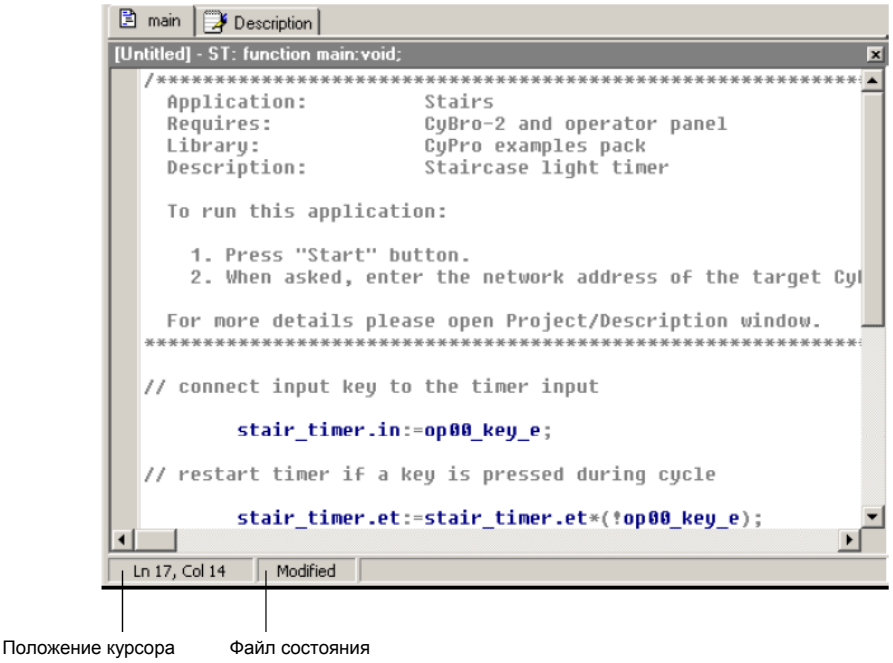


Get PLC Info	Восстановление различной информации для контроллера (PLC)
Kernel Maintenance	Загрузка нового ядра в CyBro
Environment Options	Изменение конфигурационных опций окружающей среды CyPro
Communication Monitor	Опциональный, инструментальное средство A-bus монитора, доступен с паролем

Окно редактирования

Окно редактирования используется для ввода и редактирования программы для контроллера (PLC). Каждое окно представляет одну функцию.

Команды могут записываться как доступными языками, списком команд, так и структурированным текстом. Они не могут смешиваться в одном и том же окне. Для изменения текущего языка выберите Edit/Properties (редактирование/свойства).



Содержимое редактора динамически синтаксически выделяется. Переменные, константы, функции и другие языковые элементы показываются различными цветами. Для выбора схемы различного цвета или цветов по заказу клиента используют Tools/Environment Options/Colors (инструменты/опции окружающей среды/цвета).

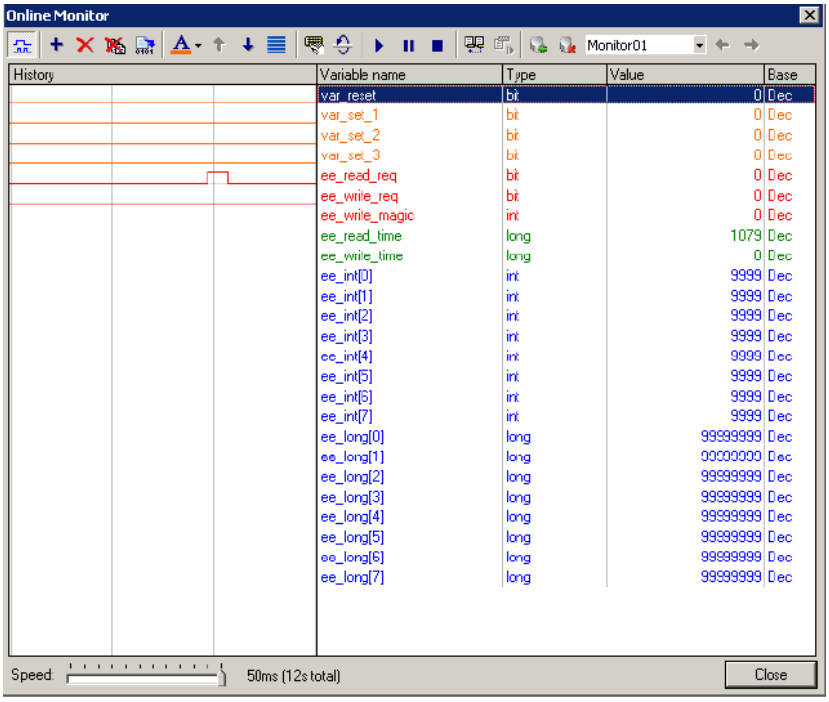
Программный помощник является полезной деталью редактора структурированного текста. Для показа списка выделенных переменных и доступных функций нажмите Ctrl-Enter.

Редактор CyPro дает возможность многократно изменять операцию. Удаление или изменение текста может сохраняться нажатием комбинации клавиш Ctrl-Z даже после других выполненных операций. Количество действий по редактированию, которые не могут не делаться, ограничивается только размером доступной памяти, которой обычно вполне достаточно.

Для понимания структуры файла .cyp откройте файл примера.

Оперативный монитор

Оперативный монитор является диалоговым окном, созданным для показа текущих значений переменных CyBro. Для его открытия выберите Program/Online Monitor (программа / оперативный монитор), кликните на значке Monitor (монитор) на коммуникационной панели инструментальных средств или нажмите клавишу F10.

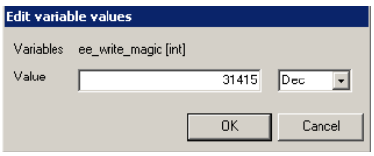


Для вставки переменных кликните на кнопке Add (добавить) и нажмите клавишу Insert (вставить). Появиться диалоговое окно Variable Insert (вставка переменной). Выберите желаемые переменные и нажмите OK. Для выбора непрерывного блока переменных нажмите кнопку Shift. При нажатии клавиши Control есть возможность сделать множественный выбор. Для трансформирования списка выберите переменную и кликните на Move Up (смещение верх) или Move Down (смещение вниз) или нажмите Control и переместите переменную клавишами (стрелками) Up или Down.

Монитор параметра допускает неограниченное число наборов. Для быстрого доступа к первым пяти нажмите комбинацию от Alt-1 до Alt-5.

Величины на мониторе обновляются примерно каждые 100 мсек. Скорость прокрутки может изменяться с использованием ползунка Speed (скорость).

Для изменения значения переменной дважды кликните по ней, нажмите кнопку Edit Variable properties (редактирование свойств переменной) или выберите опцию Properties (свойства) в меню при нажатии правой кнопки мыши.

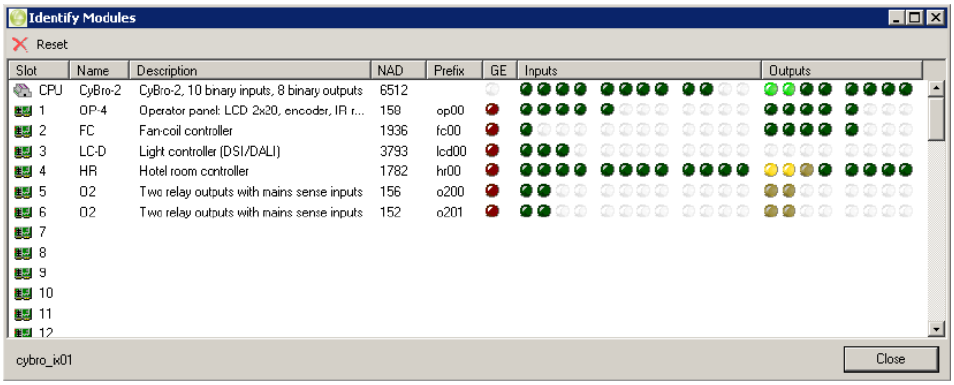


Введите новое значение и нажмите OK. введенное значение посылается в CyBro и затем считывается назад, так что монитор показывает действительное значение. Заметим, что если CyBro работает и программа изменяет переменную, ее значение будет немедленно переписываться.

Для переключения значения битовой переменной нажмите клавишу Space (пробел).

Распознавание модулей

Распознавание модулей является инструментом, используемым для идентификации модулей IEX и отдельных входов/выходов. Поскольку это операция изменения/переустановки, инструмент может использоваться одним выбранным оператором.



Каждый СИД представляет собой один цифровой вход или выход. Когда мышь позиционируется над СИД, название сигнала показывается в нижнем левом углу.

Цвета СИД входа/выхода определяются в соответствии со следующей таблицей:

СИД	Текущее значение	Измененное
	0	нет
	1	нет
	0	да
	1	да

Условие общей ошибки (GE) определяется как:

СИД	Описание
	Модуль работает правильно
	Ошибка, модуль не работает

В случае общей ошибки используйте оперативный монитор для определения причины проблемы (ошибка из-за простоя модуля, ошибка из-за шины или программная ошибка).

Чтобы идентифицировать неизвестный вход:

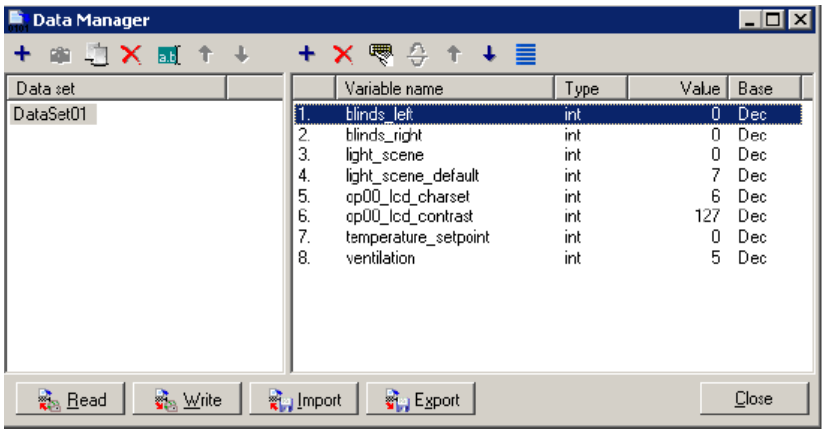
1. Переустановите “Identify Modules” (распознавание модулей).
2. Перейдите к неизвестному переключателю и нажмите на секунду.
3. Возвратитесь назад и поищите желтый СИД.

Чтобы идентифицировать неизвестный выход:

1. Кликните на СИДе выхода и проверьте активацию выходов.

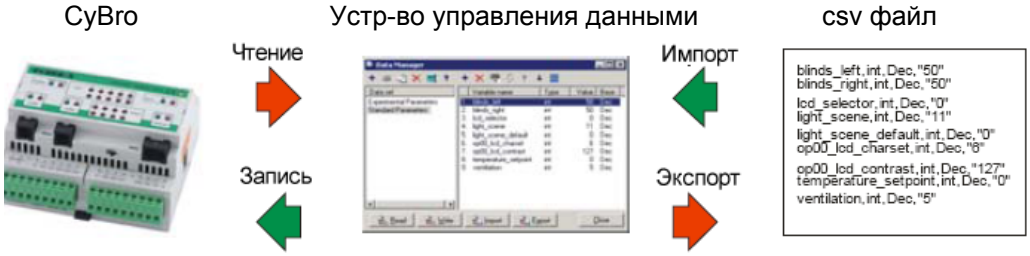
Устройство управления данными

Устройство управления данными является инструментом, используемым для передачи данных между контроллером (PLC) и ПК. Это позволяет пользователю сохранять конфигурацию контроллера (PLC) или экспортировать данные для статистического или графического анализа. Переменные организуются в наборы данных. Набор данных представляет собой определенный пользователем список переменных контроллера (PLC). Он содержит переменные и их соответствующие значения.



Доступны следующие команды:

Read (чтение)	Считывание переменных из контроллера (PLC) в устройство управления данными
Write (запись)	Запись переменных из устройства управления данными в контроллер (PLC).
Import (импорт)	Импорт переменных из csv файла в устройство управления данными.
Export (экспорт)	Экспорт переменных из устройства управления данными в csv файл.



Файл величин разделенных запятой (или csv) является только текстовым файлом, который сохраняет табличные данные. Каждая строка содержит название переменной, тип, система счисления и значение.

```
blinds left,int,Dec,"50"
blinds right,int,Dec,"50"
light scene,int,Dec,"11"
light scene default,int,Dec,"0"
op00 lcd charset,int,Dec,"6"
op00 lcd contrast,int,Dec,"127"
temperature setpoint,int,Dec,"0"
ventilation,int,Dec,"5"
```

Одинарный csv файл содержит один набор данных. Имя файла совпадает с именем набора данных. Csv формат поддерживается виртуально всеми программами для крупноформатных таблиц и баз данных. Число переменных в наборе данных не ограничивается. Для большого числа переменных чтение и запись может происходить несколько секунд.

Для открытия набора переменных контроллера (PLC) в MS Excel сделайте следующее:

- Создайте New (новый) набор данных
- Add (добавьте) переменные в набор данных
- Read (считайте) значения из контроллера (PLC)
- Export (экспортируйте) в csv файл
- Open (откройте) MS Excel и import (импортируйте) csv файл

Подобную процедуру можно использовать в противоположном направлении.

Программирование

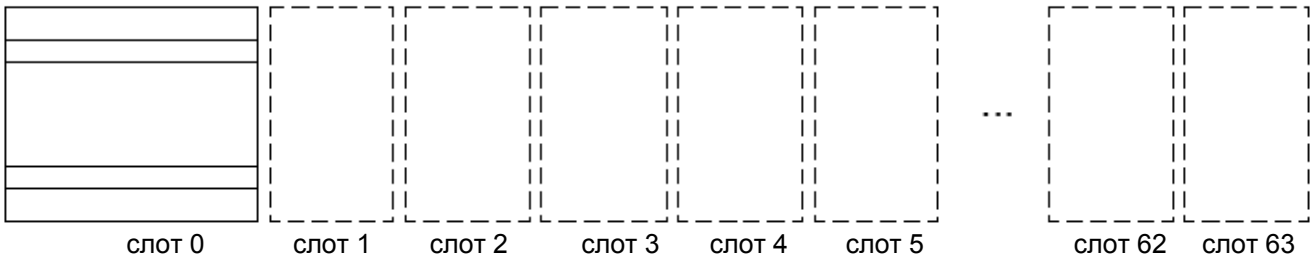
Аппаратные средства

Модули расширения

CyBro-2 может расширяться с использованием различных блоков расширения IEX-2. Полный список доступных модулей, пожалуйста, смотрите в руководстве для аппаратных средств.

Блоки расширения IEX-2

Каждый модуль IEX занимает один «слот». Слот не является физическим устройством. Он представляет собой логическое место для заполнения, используемое для адресации блока расширения.



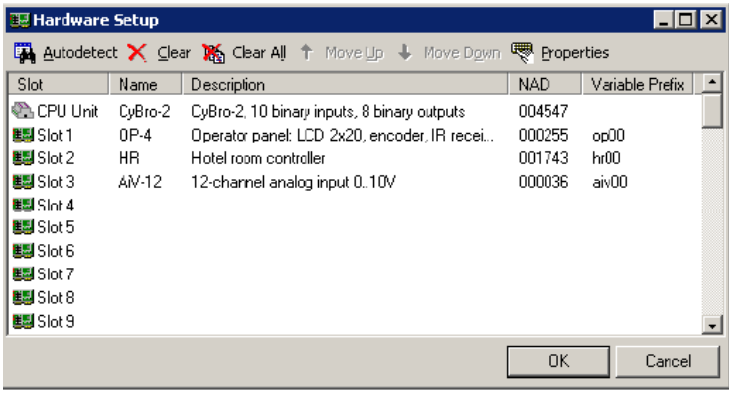
Каждый слот имеет номер от 0 до 63. Слот 0 занимает местные входы и выходы CyBro-2.

Каждый модуль IEX-2 имеет уникальный 21-битный адрес и там нет DIP переключателей. Автодетектирование будет сортировать модули в соответствии с типом и адресом в восходящем порядке. Префиксы переменной обозначаются в том же самом порядке.

Настройка аппаратных средств

Первый шаг создания проекта состоит в конфигурировании аппаратных средств.

Для выполнения автоматического детектирования аппаратных средств нажмите кнопку Autodetect (автодетектирование). Если необходимо, введите сетевой адрес соответствующего контроллера CyBro-2. После автодетектирования в диалоговом окне Hardware Setup (настройка аппаратных средств) появится конфигурация.



Детектированные CyBro-2 CPU и модули показываются в таблице.

Для идентификации сетевого адреса неизвестного CyBro введите нуль как сетевой адрес и выполните Autodetect (автодетектирование). Блок CPU и NAD (адрес) будет появляться в первой строке в соответствии с детектированными платами IEX. При использовании более одного присоединенного контроллера (PLC) каждый NAD (адрес) CyBro должен вводиться вручную.

Переменные

Обозначение имен

Имя переменной может состоять из любого набора, включающего буквы, цифры и подстрочные знаки. При том, что первый символ не цифра. Максимальная длина 24 символа. Имена не чувствительны к написанию, так “Valve” и “valve” являются одинаковыми переменными. Специальные национальные символы не поддерживаются.

Примеры допустимых имен:

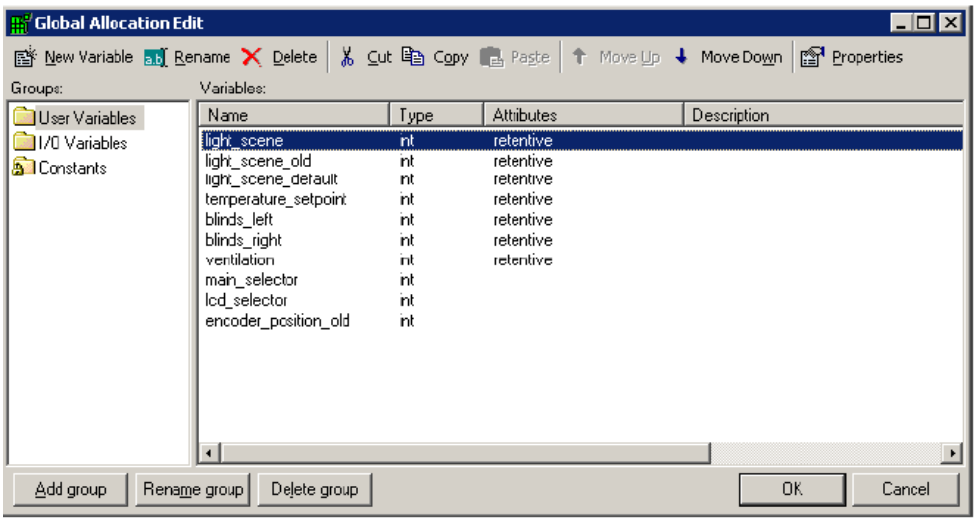
cnt
track5a
caret_position
valve_open_contact
MaximumWaterLevel

Имя переменной не совпадает с любым паролем IEC-1131-3.

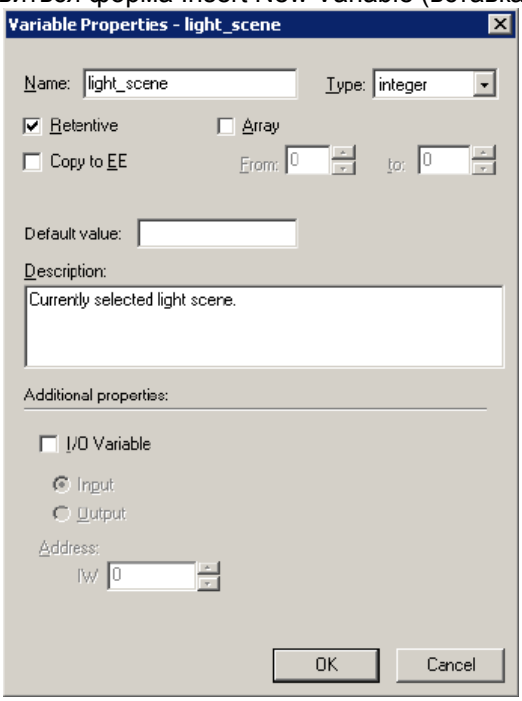
Выделение памяти

В соответствии с порядком функционирования IEC-1131-3 память не может принять адрес. Каждая переменная должна иметь уникальное имя и строго определенный тип. То же самое относится к входам и выходам.

Память для переменных выделяется с использованием Global Allocation Edit (глобальное редактирование выделения памяти):



Для вставки новой переменной выберите группу, где будет храниться новая переменная и выберите New Variable (новая переменная). Появиться форма Insert New Variable (вставка новой переменной).



Должно вводиться имя переменной. Если имя неправильное, кнопка OK недоступна.

Сводка типов данных:

тип	размер (биты)	диапазон
bit	1	0..1
integer	2	-32768..32767
longint	4	-2147483648..2147483647
real	4	-1e38..1e38
word	16	-

Bit, integer, longint и real являются основным типом данным.

Bit является единственной логической переменной с только двумя возможными значениями, нуль и единица. Он должен использоваться для признаков, логических уравнений и подобных типов. In bit и out bit оба относятся к битовому типу. Значение команд сравнения 0 также относится к битовому типу.

Integer (целое число) является 16-битным именем. Он должен использоваться для подсчета, состояния кодирования, фиксированной арифметической точки и подобных типов.

Longint (длинной целое число) является 32-битным именем. Он должен использоваться вместо целого числа, когда ожидаемые величины выше, чем 32 767.

Real (реальное число) относится к величинам с плавающей точкой. Команды для плавающей точки обычно используют в три раза больше памяти по сравнению с целыми числами и обработка чисел с плавающей точкой происходит примерно в десять раз медленнее, чем для целых чисел.

Word (слово) является массивом из 16 бит. Оно не представляет величины – она только указывает метод выполнения логических операций до 16 бит одновременно. Арифметические операции не применяются для словесных переменных.

Переменные in bit, out bit, in word и out word представляют собой физически присоединенные бинарные (бит) и аналоговые (целочисленные) сигналы.

Устройства (таймеры и счетчики) структурируются по типам данных, состоящим из нескольких специальных областей деятельности.

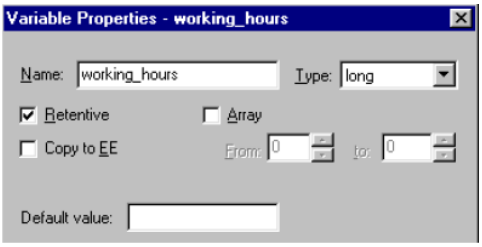
Константы используются для представления значения, которое не будет изменяться. Как пример, для тригонометрический вычислений может определяться значение Pi=3,14. Для констант не используются типы данных.

Также доступны команды преобразования типа, но должны быть причины для их использования, поскольку они вероятно могут быть причиной ошибок. Ограниченное количество преобразований типа указывает на правильность разработки и хороший стиль программирования.

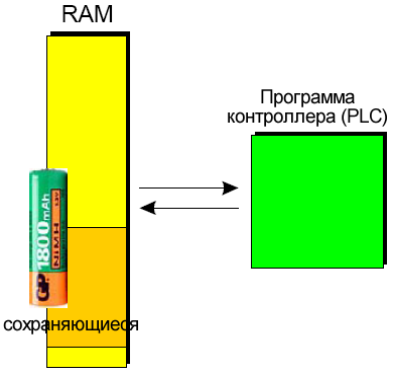
Сохраняющиеся переменные

Сохраняющиеся переменные остаются на их значениях, когда источник питания выключается, а также, когда контроллер (PLC) останавливается/запускается.

Чтобы сделать переменную сохраняющейся, установите признак сохранения в глобальный диалог выделения памяти. Признак сохранения устанавливается для каждой переменной индивидуально.



Поскольку как сохраняющиеся, так и несохраняющиеся переменные физически хранятся в RAM (оперативная память), сохраняющиеся переменные используются в программе контроллера (PLC) точно таким же образом, как несохраняющиеся. Единственная разница состоит в том, что несохраняющиеся переменные удаляются системным программным обеспечением во время подачи питания и запуска программы.



Число сохраняющихся переменных не ограничивается. Если требуется, вся память контроллера (PLC) может быть использована под сохраняющиеся переменные.

Время сохранения данных сохраняющихся переменных указывается в руководстве для аппаратных средств CyBro. При отключении питания на время более указанного, содержание памяти сохраняющихся переменных может быть утеряно.

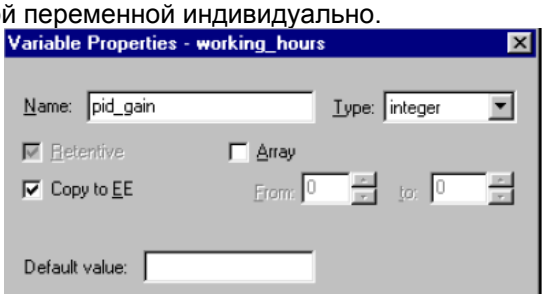
Системный бит retentive_fail указывает, что память сохраняющихся переменных утеряна. Она настраивается автоматически после включения питания и удаляется каждый раз при запуске контроллера (PLC). Заметим, что retentive_fail указывает на то, что память для сохраняющихся переменных точно утеряна. Если есть только частичная потеря памяти, она может показываться или может не показываться. Для гарантирования 100% защиты памяти пользователь должен обеспечить его собственную защиту контрольной суммы/crc.

Пересылка новой программы в контроллер (PLC) будет приводить к удалению всех переменных, включая сохраняющиеся. Сохраняющиеся переменные будут сохраняться только, если список выделения памяти не изменяется. Для пересылки программы без удаления переменных используют команду Send Without Init (пересылка без инициализации).

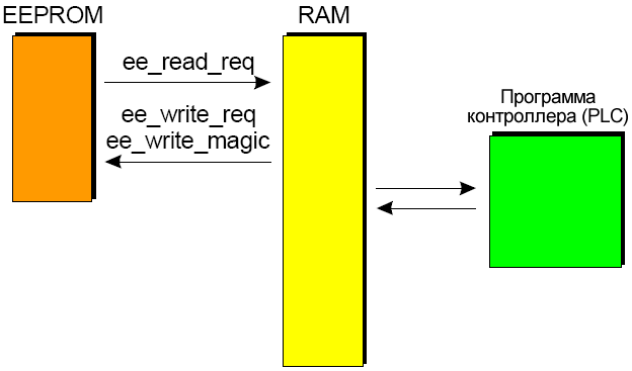
EE переменные

Переменные, которые нуждаются в сохранении длительное время, должны храниться в EEPROM (электрически стираемая память) памяти.

Чтобы сделать долговременную переменную, установите копирование признака EE (внешнее оборудование) в глобальный диалог выделения памяти. Копирование признака EE (внешнее оборудование) должно определяться для каждой переменной индивидуально.



Переменные EE (внешнее оборудование) постоянно хранятся в RAM памяти, как все другие сохраняемые и несохраняемые переменные, но они также имеют копию в EEPROM (электрически стираемая память). Из-за этого они используются программой контроллера (PLC) точно также как другие переменные, но кроме того, доступно чтение и запись EE (внешнее оборудование).



Для чтения всех переменных их EE (внешнее оборудование) в RAM настройте бит ee_read_req. Бит будет автоматически удаляться при окончании копирования. В зависимости от числа переменных EE (внешнее оборудование), процесс копирования может продолжаться от нескольких миллисекунд до 3-4 секунд.

Для записи всех переменных из RAM в EE (внешнее оборудование) установите ee_write_magic на 31415 и настройте ee_write_req. При окончании копирования обе переменных будут удалены. В зависимости от числа переменных EE (внешнее оборудование) процесс записи может продолжаться от нескольких миллисекунд до 5-6 секунд. В результате необходимо защитить память EE (внешнее оборудование) от случайной записи.

Для проверки работы системы откройте пример программы EEReadWriteDemo.cyp.

Только вся память EE (внешнее оборудование) может читаться или записываться и, при этом, нет метода чтения и записи одной переменной.

Переменные EE (внешнее оборудование) не должны быть доступны для программы контроллера (PLC) во время операции, как чтения, так и записи. Чтение и запись не активируется одновременно. Удаление битов ee_read_req и ee_write_req во время чтения или записи может привести к неожиданным результатом и этого нужно избегать.

Все переменные EE (внешнее оборудование) автоматически считываются при включении питании. Переменные могут использоваться, когда заканчивается считывание, после перехода на нуль ee_read_req.

Важные или чувствительные переменные также могут храниться в EE (внешнее оборудование). Память EE (внешнее оборудование) является намного более безопасным местом по сравнению со случайным повреждением по причине электромагнитных скачков. Если содержимое RAM повреждается, есть все шансы, что содержание EEPROM (электрически стираемая память) сохранится.

Полное число переменных EE (внешнее оборудование) ограничивается физическим размером памяти и указывается в руководстве для аппаратных средств. Для проверки используемой памяти откройте диалоговое окно PLC Info, таблицу PLC Program (программа контроллера), Total EE size (полный размер внешнее оборудование).

Время сохранения в EEPROM (электрически стираемая память) 100 лет. Ресурс работы около 1 миллиона циклов записи.

Переменные Вх/Вых

Переменные Вх/Вых используются для доступа к физическим входам и выходам. Автоматическая настройка аппаратного оборудования выделяет переменные Вх/Вых.

Хотя возможно изменить имя переменной, советуем использовать имена по умолчанию. Если настройка аппаратных средств изменяется, автодетектирование не может правильно расположить переименованные переменные.

CyBro-2 использует четыре адресных пространства для доступа к Вх/Вых, два двоичных и два аналоговых:

IX	Входной бит
QX	Выходной бит
IW	Входное слово
QW	Выходное слово

Для двоичных входов и выходов выделяется место, соответственно начиная с ix0, как для первого физического входа и qx0, как для первого физического выхода. Остающееся место резервируется для плат расширения.

Двоичные входы

IX2047 IX2016	слот 63
…	
IX63 IX32	слот 1
IX31 IX0	на плате

Двоичные выходы

QX2047 QX2016	слот 63
…	
QX63 QX32	слот 1
QX31 QX0	на плате

Аналоговые платы Вх/Вых имеют 32 слова, зарезервированные для каждого слота. Одноканальные платы используют только первое зарезервированное слово. Слот 0 резервируется, так что первым доступным слотом является 1. Переменные in word (вход слова) и out word (выход слова) оба относятся к целочисленному типу.

Аналоговые входы

IW2047 IW2016	слот 63
…	
IW63 IW32	слот 1
IW31 IW0	резерв

Аналоговые выходы

QW2047 QW2016	слот 63
…	
QW63 QW32	слот 1
QW31 QW0	резерв

Переменные входа и выхода являются автовыделяемыми и их имена имеют вид:

nnnxx_varname

где nnn это имя типа модуля (например, bio для Bio-24, op для OP-2, xx это порядковый номер модуля, начинающийся с 00 (т.е. третья панель оператора будет 02) и varname является именем переменной.

Например, переменная Вх/Вых key_f отнесенная к кнопке F первой панели оператора будет обозначаться как op00_key_f.

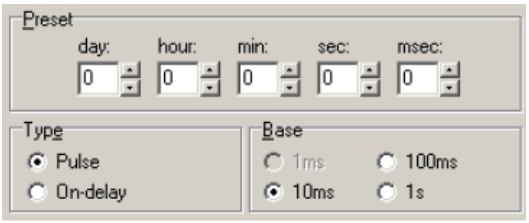
Внутренние переменные

Внутренние переменные CyBro-2 располагаются в адресном пространстве Вх/Вых. Каждый параметр имеет специфическую функцию.

first_scan	Активен только во время первого скана. Может использоваться для инициализации переменных.
scan_overrun	Указывает, что скан вышел за предел установленного времени. Если программный цикл длился больше 50 мсек, выполнение будет прерываться.
clock_10ms	10 мсек системного времени, 5 мсек выше и 5 мсек ниже.
clock_100ms	100 мсек системного времени, 50 мсек выше и 50 мсек ниже.
clock_1s	Одна секунда системного времени, 500 мсек выше и 500 мсек ниже.
clock_10s	Десять секунд системного времени, 5 сек выше и 5 сек ниже.
clock_1min	Одна минута системного времени, 30 сек выше и 30 сек ниже.
retentive_fail	Указывает, что сохраняющаяся память не имеет силу, поскольку питание было выключено слишком долгое время.
all_output_off	Если активно, все двоичные выходы CyBro будут немедленно перестают работают.
all_input_off	Если активно, все двоичные входы CyBro отсоединяются, оставляя последнее значение.
no_input_filter	Если активно, выключается мастный 5 мсек входной фильтр
rtc_write_req	Настройка записи текущего времени/даты в часах реального времени (RTC). При окончании будет происходить автоматическая переустановка.
ee_read_req	Настройка чтения всех долговременных переменных из EEPROM. Она автоматически происходит во время включения питания. При окончании будет происходить автоматическая переустановка.
ee_write_req	Настройка записи всех долговременных переменных в EEPROM. При окончании будет происходить автоматическая переустановка.
ee_write_magic	Используется как защита записи EEPROM. Настройка на 31415 делает доступным запись. При окончании записи будет происходить автоматическая переустановка.
scan_time	Время выполнения последнего скана в миллисекундах.
scan_time_max	Максимальное ограничение времени выполнения скана.
scan_frequency	Число сканов за секунду. Нуль, если программа остановлена.
rtc_sec rtc_min rtc_hour rtc_weekday rtc_date rtc_month rtc_year	Данные часов реального времени.

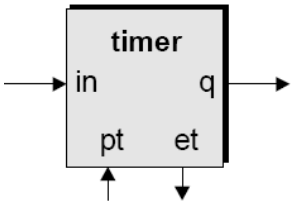
Таймеры

Таймеры относятся к специальным типам структурированных переменных, используемым для определения временного интервала. Для определения новой переменной таймера откройте диалоговое окно Insert New Variable (вставка новой переменной), выберите тип таймера, введите имя, настройте предустановку, тип и базу таймера и нажмите ОК.



База таймера является периодом, на который таймер имеет приращение, т.е. временное разрешение таймера. База должна быть равна или больше, чем программное время сканирования. Поэтому база в 10 мсек не подходит для длинной программы CyBro.

Таймер может представляться как функциональный блок с двумя входами и двумя выходами:



Подобным образом переменная таймера состоит из четырех полей. Каждое поле является примитивным типом данных. Полями являются:

имя	направление	тип	описание
in	вход	бит	вход
q	выход	бит	выход
pt	вход	длительный	предустановленное время
et	выход	длительный	пройденное время

Для работы с таймером из программы CyBro используйте следующий синтаксис:

```
<timer name> . <field>
```

Например, для настройки предустановки wash_timer на 15 секунд (предполагается, что база 100 мсек):

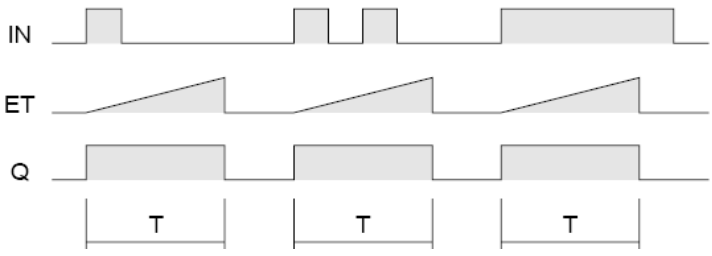
```
wash_timer . pt : = 150
```

Пройденное время wash_timer будет начинаться с 0 и будет иметь приращение каждые 100 мсек до тех пор, пока оно не достигнет 150.

Импульсный таймер

Выход таймера немедленно активируется после переднего фронта входного сигнала. После указанного времени выход будет прерываться. Изменения входного сигнала во время импульса не оказывает воздействие на выход.

Рисунок показывает типичную работу импульсного таймера:

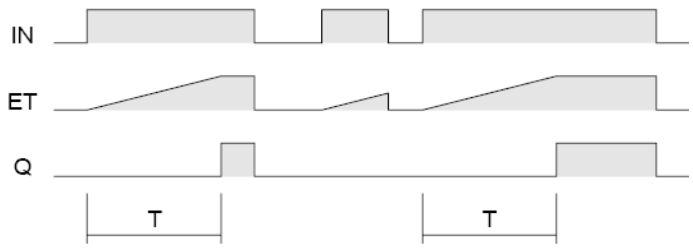


Типичным применением является ступенчатый индикаторный таймер.

Реле времени

При активации входа таймер начинает отсчет. После указанного времени активируется выход и остается на верхнем уровне до тех пор, пока вход остается на низком уровне. Доступными полями являются те же, что и для импульсного таймера.

Рисунок показывает типичную работу реле времени:

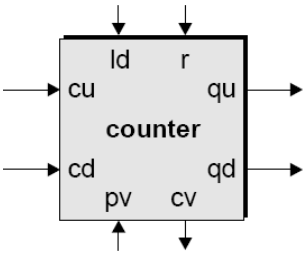


Типичным применением является переключатель звезда-треугольник для разгоняемых трехфазных двигателей.

Счетчики

Передний фронт на входе счетчика увеличивает (или уменьшает) значение на счетчике на единицу. Когда значение достигает предустановленного предела, активируется выход счетчика. Каждое изменение мгновенно, так что немедленно после этого выход может использоваться.

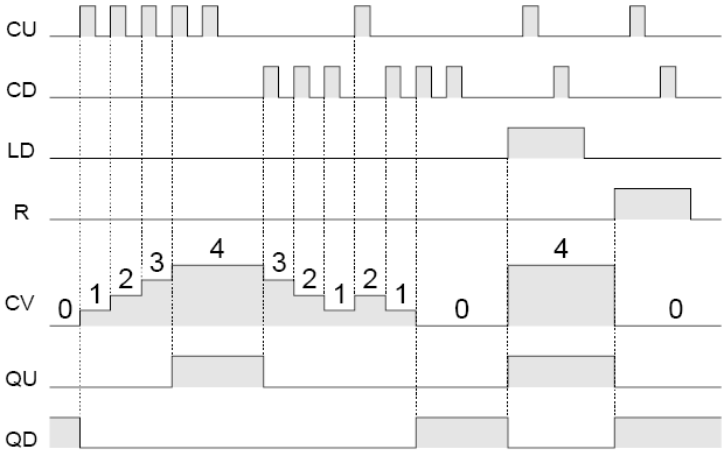
Счетчик может представляться как функциональный блок с пятью входами и тремя выходами:



Доступные поля для счетчика:

имя	направление	тип	описание
cu	вход	бит	счет вверх
cd	вход	бит	счет вниз
ld	вход	бит	нагрузка
r	вход	бит	переустановка
pv	вход	целое число	предустановленное значение
cv	выход	целое число	значение на счетчике
qu	выход	бит	верхний предел
qd	выход	бит	нижний предел

Рисунок ниже показывает типичную работу счетчика (предустановленное значение = 4):



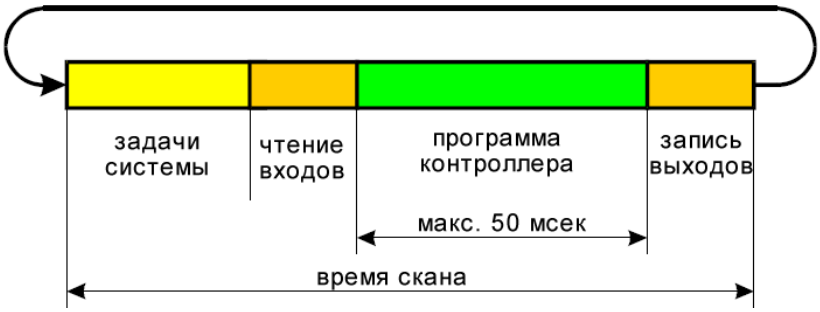
Процесс обновления

CyBro выполняет обновляет систему обработки – немедленно перед этим входы тестируются и выходы немедленно обновляются после выполнения задачи. Даже если вход изменяется во время выполнения задачи, значение входа остается стабильным во время скана.

Хотя выполнение обновления происходит достаточно медленно, оно делает программирование более легким.

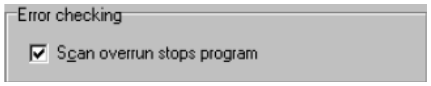
Сканирование переполнения

Время сканирования определяется как время необходимое для выполнения одного цикла программы (скана). Цикл программы состоит из обработки данных системы и программы контроллера (PLC) (пользователя).



Если время выполнения программы контроллера (PLC) превышает 50 мсек, CyBro вводит состояние ошибки переполнения сканирования и останавливает выполнение программы. Код ошибки показывается на строке состояния.

В окончательной версии программы контроллера (PLC) обычно недоступна проверка ошибки переполнения сканирования. Чтобы сделать это, отменяют проверку позиции для отметки Scan overrun stops program (остановка программы переполнения сканирования) расположенной в диалоговом окне Program Properties (свойства программы), закладка PLC (контроллер).



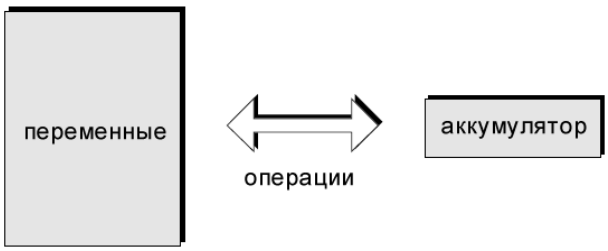
В этом случае программа размером более 50 мсек будет прерываться и запускаться сначала. Будет настраиваться системная переменная scan_overrun, но программа продолжает работу без ошибки.

Список команд

Список команд является языком низкого уровня со структурой подобной языку ассемблера. Каждая строка из кода списка инструкций состоит из четырех частей: метка, команда, операнд и комментарий.

```
cnt_beg:    ld      product_count    // counting products
   |         |           |                  |
метка     команда     операнд          комментарий
```

Модель вычисления программы списка команд состоит из переменной, для которой выделена память, и сумматор. Все арифметические и логические операторы выполняются аккумулятором.



В отличие от классического ассемблера, аккумулятор может содержать величину любого типа. Тип определяется операндом. Если тип операнда не уникальный, тип определяется из последовательных команд. Панель отслеживания типа показывает тип для каждой строки.

Аккумулятор загружается однократно, тип не может изменяться до тех пор, пока результат операции не сохранится. Исключением являются команды сравнения и команды для изменения типа.

Команды одного операнда обычно использует аккумулятор как операнд. В некоторых случаях, возможно, выполнить команду прямо на переменной.

Аккумулятор является всегда первым операндом в командах с двумя операндами. Второй операнд может быть постоянным или переменным. Результат операции также хранится в аккумуляторе.

Типичная арифметическая или логическая последовательность должна загружать величину в аккумулятор (шаг 1), выполнение операции (шаг 2) и сохранение результата (шаг 3).

Каждая команда должна записываться в отдельной строке.

Например, для приращения переменной product_count необходима следующая последовательность:

```
ld      product_count
add     1
st      product_count
```

Значение загружается в аккумулятор, приращивается и затем сохраняется.

Список команд допускает два стиля комментирования. Одна строка комментария начинается с двух слэшей:

```
                ld              valve           //  hot water
```

Многострочный комментарий ограничивается в пределах пары /**/:

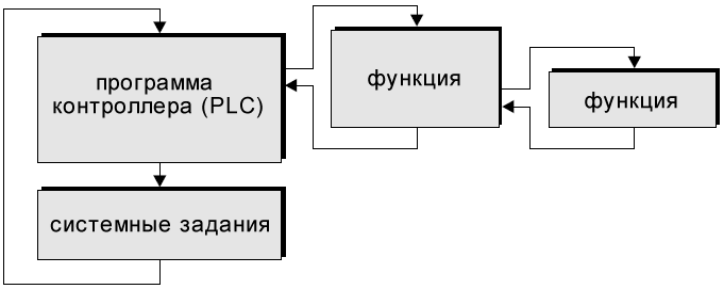
```
/*****************************************************************************
                        function move caret ( )
*****************************************************************************
    input:          caret position: integer
    output:             caret done: bit
                    caret error code: integer
    description:    move caret until caret position is reached
*****************************************************************************/
```

Метки используются для маркировки адреса назначения команды перехода. Метка должна записываться в первой колонке и за ним должно следовать двоеточие:

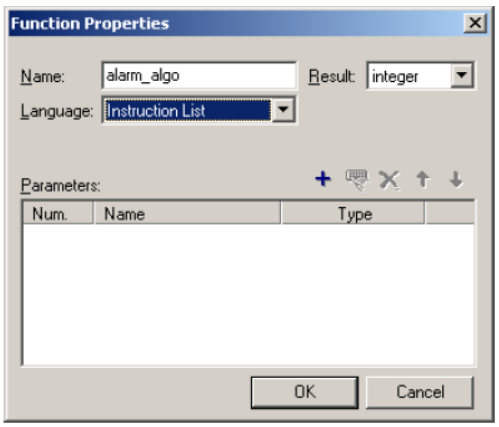
```
                jmp             block end

                ld              test value
                add             1
                st              test value

block_end:
```

Команды перехода должны использоваться осторожно. Безусловный переход назад может привести к бесконечному циклу, который может быть причиной ошибки из-за увеличения темпа скана.

Большие программы могут разделяться на функции. Функция содержит кодовый сегмент, который может вызываться из основного задания или из другой функции. Уровень вложенности ограничивается только размером доступной памяти.



Для создания новой функции кликните правой кнопкой мыши на родительской функции и выберите Add (добавить). Появится форма Function Properties (свойства функции). Введите Name (имя), выберите Language (язык) и нажмите **Enter** (ввод).



Функция вызывается командой cal.

```
cal alarm_algo
```

Функция может вызываться много раз из различных мест, но такая функция не должна содержать команд fp или fn. Функции могут записываться различными языками.

Загрузка и хранение команд

команда:	LD
операция:	ACC := arg

bit	int	word	long	real	acc	const	var
•	•	•	•	•		•	•
изменение типа на:					нет изменения		

Загрузка значения константы и переменной в аккумулятор.

команда:	LDN
операция:	ACC := !arg

bit	int	word	long	real	acc	const	var
•						•	•
изменение типа на:					нет изменения		

Загрузка значения переменной инверсного бита в аккумулятор.

команда:	ST
операция:	var := ACC

bit	int	word	long	real	acc	const	var
•	•	•	•	•			•
изменение типа на:					нет изменения		

Загрузка значения аккумулятора в указанную переменную.

команда:	STN
операция:	var := !ACC

bit	int	word	long	real	acc	const	var
•							•
изменение типа на:					нет изменения		

Загрузка инверсного значения аккумулятора в указанную переменную.

команда:	SET
операция:	arg := 1

bit	int	word	long	real	acc	const	var
•					•		•
изменение типа на:					нет, если исп. var		

Установка аргумента в логическое условие TRUE (истинное).

команда:	RES
операция:	arg := 0

bit	int	word	long	real	acc	const	var
•					•		•
изменение типа на:					нет, если исп. var		

Удаление аргумента в логическом условии FALSE (ошибочное).

команда:	SETC
операция:	var := ACC | var

bit	int	word	long	real	acc	const	var
•							•
изменение типа на:					нет изменения		

Настройка переменной, если аккумулятор истинный, в другом случае ничего не делается.

команда:	RESC
операция:	var := !ACC & var

bit	int	word	long	real	acc	const	var
•							•
тип изменение на:					нет изменения		

Удаление переменной, если аккумулятор истинный, в другом случае ничего не делается.

Логические команды

команда:	AND
операция:	ACC := ACC & arg

bit	int	word	long	real	acc	const	var
•		•				•	•
изменение типа на:						нет изменения	

Выполнение логического AND (и) с аккумулятором и аргументом и сохранение результата в аккумуляторе.

ACC перед	arg	ACC после
0	0	0
0	1	0
1	0	0
1	1	1

команда:	ANDN
операция:	ACC := ACC & !arg

bit	int	word	long	real	acc	const	var
•		•				•	•
изменение типа на:						нет изменения	

Выполнение логического AND (и) с аккумулятором и инверсным аргументом и сохранение результата в аккумуляторе.

ACC перед	arg	ACC после
0	0	0
0	1	0
1	0	1
1	1	0

команда:	OR
операция:	ACC := ACC | arg

bit	int	word	long	real	acc	const	var
•		•				•	•
изменение типа на:						нет изменения	

Выполнение логического OR (или) с аккумулятором и аргументом и сохранение результата в аккумуляторе.

ACC перед	arg	ACC после
0	0	0
0	1	1
1	0	1
1	1	1

команда:	OR
операция:	ACC := ACC | !arg

bit	int	word	long	real	acc	const	var
•		•				•	•
изменение типа на:						нет изменения	

Выполнение логического OR (или) с аккумулятором и инверсным аргументом и сохранение результата в аккумуляторе.

ACC перед	arg	ACC после
0	0	1
0	1	0
1	0	1
1	1	1

команда:	XOR
операция:	ACC := ACC ^ arg

bit	int	word	long	real	acc	const	var
•		•				•	•
изменение типа на:					нет изменения		

Выполнение логического XOR (исключающее или) с аккумулятором и аргументом и сохранение результата в аккумуляторе.

ACC перед	arg	ACC после
0	0	0
0	1	1
1	0	1
1	1	0

команда:	XORN
операция:	ACC := ACC ^ !arg

bit	int	word	long	real	acc	const	var
•		•				•	•
изменение типа на:					нет изменения		

Выполнение логического XOR (исключающее или) с аккумулятором и инверсным аргументом и сохранение результата в аккумуляторе.

ACC перед	arg	ACC после
0	0	1
0	1	0
1	0	0
1	1	1

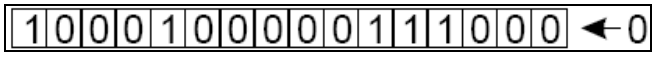
команда:	SHL
операция:	-

bit	int	word	long	real	acc	const	var
		•			•		
изменение типа на:					нет изменения		

Выполнение смещения логического бита налево. Бит минимального значения (бит 0) заполняется 0.

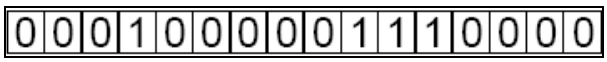
Перед операцией аккумулятор содержит:

MSB LSB



После операции аккумулятор содержит:

MSB LSB



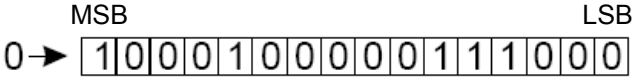
команда:	SHR
операция:	-

bit	int	word	long	real	acc	const	var
		•			•		
изменение типа на:					нет изменения		

Выполнение смещения логического бита направо. Бит максимального значения (бит 15) заполняется 0.

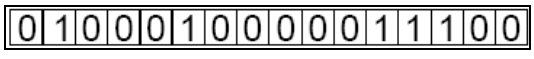
Перед операцией аккумулятор содержит:

MSB LSB



После операции аккумулятор содержит:

MSB LSB



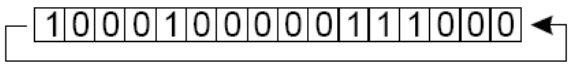
команда:	ROL
операция:	-

bit	int	word	long	real	acc	const	var
		•			•		
изменение типа на:					нет изменения		

Выполнение вращения логического бита налево. Бит максимального значения (бит 15) сдвигается, по крайней мере, на значение бита (бит 0).

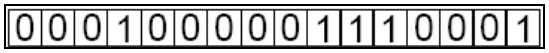
Перед операцией аккумулятор содержит:

MSB LSB



После операции аккумулятор содержит:

MSB LSB



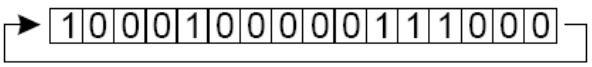
команда:	ROR
операция:	-

bit	int	word	long	real	acc	const	var
		•			•		
изменение типа на:					нет изменения		

Выполнение вращения логического бита направо. Бит минимального значения (бит 0) сдвигается на бит большего значения (бит 15).

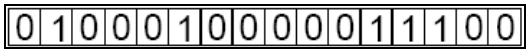
Перед операцией аккумулятор содержит:

MSB LSB



После операции аккумулятор содержит:

MSB LSB



команда:	CPL
операция:	arg := !arg

bit	int	word	long	real	acc	const	var
•					•		•
изменение типа на:					нет, если исп. var		

Дополнение значения переменной.

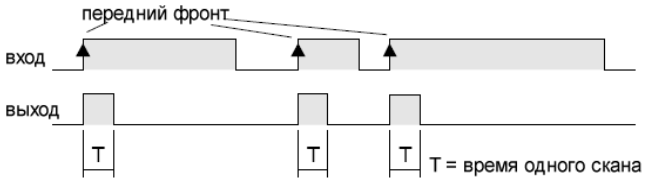
Краевые команды

команда:	FP
операция:	-

bit	int	word	long	real	acc	const	var
•					•		•
изменение типа на:					нет изменения		

Определение переднего фронта значения аккумулятора. При появлении переднего фронта аккумулятор будет условие TRUE (истинное).

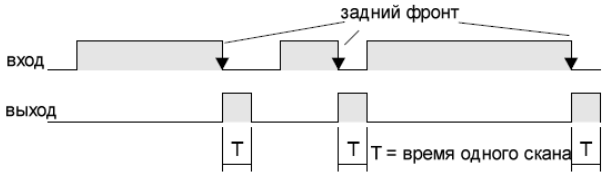
```
        fp      bit     // detect raising edge
        jmpnc   next    // jump over if edge isn't detected
        ld      cnt     // load variable cnt
        add     1       // increment by 1
        st      cnt     // store to cnt

next:
```



команда:	FN
операция:	-

bit	int	word	long	real	acc	const	var
•					•		•
изменение типа на:					нет изменения		

Определение заднего фронта значения аккумулятора. При появлении заднего фронта аккумулятор будет условие TRUE (истинное) во время одного скана.



Арифметические команды

команда:	ADD
операция:	ACC := ACC + arg

bit	int	word	long	real	acc	const	var
	•		•	•		•	•
изменение типа на:					нет изменения		

Добавление значения аргумента в аккумулятор.

команда:	SUB
операция:	ACC := ACC - arg

bit	int	word	long	real	acc	const	var
	•		•	•		•	•
изменение типа на:					нет изменения		

Вычитание значения аргумента из аккумулятора.

команда:	MUL
операция:	ACC := ACC * arg

bit	int	word	long	real	acc	const	var
	•		•	•		•	•
изменение типа на:					нет изменения		

Умножение значения аккумулятора на аргумент и сохранение результата в аккумулятор. Никакого детектирования переполнения не выполняется. Если результат превышает типичные границы, он будет отбрасываться.

команда:	DIV
операция:	ACC := ACC / arg

bit	int	word	long	real	acc	const	var
	•		•	•		•	•
изменение типа на:					нет изменения		

Деление значения аккумулятора на аргумент и сохранение результата в аккумулятор. Если значение аргумента 0, выполняется запрещенное деление и результат будет 0.

команда:	MOD
операция:	ACC := ACC % arg

bit	int	word	long	real	acc	const	var
	•		•	•		•	•
изменение типа на:					нет изменения		

Вычисление в устройстве деления и сохранение его в аккумуляторе. Если значение аргумента 0, выполняется запрещенное деление и результат будет 0.

команда:	NEG
операция:	ACC := - ACC

bit	int	word	long	real	acc	const	var
	•		•	•	•		
изменение типа на:					нет изменения		

Перевод значения аккумулятора в отрицательные значения.

Команды сравнения

Команды сравнения выполняются для сравниваемых значений данных одного и того же типа. Все команды производят возврат битового типа данных в результате в аккумулятор. TRUE (истинное), если сравнение имело успех и FALSE (ошибочное), если оно неудачно.

команда:	EG
операция:	ACC := ACC == arg

bit	int	word	long	real	acc	const	var
•	•	•	•	•		•	•
изменение типа на:					бит		

Проверьте, что аккумулятор равен аргументу и результат сохранен в аккумуляторе.

команда:	NE
операция:	ACC := ACC != arg

bit	int	word	long	real	acc	const	var
•	•	•	•	•		•	•
изменение типа на:					бит		

Проверьте, что аккумулятор не равен аргументу и результат сохранен в аккумуляторе.

команда:	GT
операция:	ACC := ACC > arg

bit	int	word	long	real	acc	const	var
	•		•	•		•	•
изменение типа на:					бит		

Проверьте, что аккумулятор больше чем аргумент и результат сохранен в аккумуляторе.

команда:	GE
операция:	ACC := ACC >= arg

bit	int	word	long	real	acc	const	var
	•		•	•		•	•
изменение типа на:					бит		

Проверьте, что аккумулятор больше или равен аргументу и результат сохранен в аккумуляторе.

команда:	LT
операция:	ACC := ACC < arg

bit	int	word	long	real	acc	const	var
	•		•	•		•	•
изменение типа на:					бит		

Проверьте, что аккумулятор меньше чем аргументу и результат сохранен в аккумуляторе.

команда:	LE
операция:	ACC := ACC <= arg

bit	int	word	long	real	acc	const	var
	•		•	•		•	•
изменение типа на:					бит		

Проверьте, что значение в аккумуляторе меньше или равен аргументу и результат сохранен в аккумуляторе.

Типовые команды на преобразование

Поскольку типы данных используемых в CyBro определяются очень строго, есть ряд типовых команд на преобразование. Нижние типы ряда могут преобразовываться в более высокие типы ряда без риска любой потери данных, но не случае, когда идет преобразование от более высокого к более низкому типу ряда (например, длинное целое число к целому числу). Во втором случае успешное преобразование будет зависеть от величины преобразования.

команда:	BTOI
операция:	-

bit	int	word	long	real	acc	const	var
•					•		
изменение типа на:					целое число		

Преобразование бита в целое число.

команда:	BTOL
операция:	-

bit	int	word	long	real	acc	const	var
•					•		
изменение типа на:					дл. целое число		

Преобразование бита в длинное целое число.

команда:	BTOW
операция:	-

bit	int	word	long	real	acc	const	var
•					•		
изменение типа на:					слово		

Преобразование бита в слово.

команда:	ITOL
операция:	-

bit	int	word	long	real	acc	const	var
	•				•		
изменение типа на:					дл. целое число		

Преобразование целого числа в длинное целое число.

команда:	ITOR
операция:	-

bit	int	word	long	real	acc	const	var
	•				•		
изменение типа на:					реальное число		

Преобразование целого числа в реальное число.

команда:	ITOW
операция:	-

bit	int	word	long	real	acc	const	var
	•				•		
изменение типа на:					слово		

Преобразование целого числа в слово.

команда:	LTOI
операция:	-

bit	int	word	long	real	acc	const	var
			•		•		
изменение типа на:					целое число		

Преобразование длинного целого числа в целое число. Заметим, что значение аккумулятора может быть усечено, если оно превышает границы целого числа.

команда:	LTOR
операция:	-

bit	int	word	long	real	acc	const	var
			•		•		
изменение типа на:					реальное число		

Преобразование длинного целого числа в реальное число.

команда:	LTOW
операция:	-

bit	int	word	long	real	acc	const	var
			•		•		
изменение типа на:					слово		

Преобразование длинного целого числа в слово. Заметим, что значение аккумулятора может быть усечено, если оно превышает границы слова.

команда:	RTOI
операция:	-

bit	int	word	long	real	acc	const	var
				•	•		
изменение типа на:					целое число		

Округление реального числа и преобразование в целое число. Заметим, что значение аккумулятора может быть усечено, если оно превышает границы целого числа.

команда:	RTOL
операция:	-

bit	int	word	long	real	acc	const	var
				•	•		
изменение типа на:					дл. целое число		

Округление реального числа и преобразование в длинное целое число. Заметим, что значение аккумулятора может быть усечено, если оно превышает границы длинного целого числа.

команда:	WTOI
операция:	-

bit	int	word	long	real	acc	const	var
		•			•		
изменение типа на:					целое число		

Преобразование слова в целое число.

команда:	WTOL
операция:	-

bit	int	word	long	real	acc	const	var
		•			•		
изменение типа на:					дл. целое число		

Преобразование слова в длинное целое число.

Команды ветвления

Команда ветвления определяет порядок выполнения программы. Условные команды ветвления выполняются только, если аккумулятор содержит значение TRUE (истинное), в то время как безусловные команды ветвления выполняются в каждом скане. Прыжковые команды должны использоваться осторожно, поскольку возможен бесконечный цикл, который может быть причиной возврата в исходное положение. Также круговой вызов функций может быть причиной той же самой ошибки (например, sub1 вызывает sub2, которое опять вызывает sub1).

команда:	JMP label

bit	int	word	long	real	acc	const	var
изменение типа на:					ничто		

Перескок безусловно на метку.

команда:	JMPC label

bit	int	word	long	real	acc	const	var
•							
изменение типа на:					ничто		

Перескок условно на метку, если значение аккумулятора устанавливается на TRUE (истинное).

команда:	JMPNC label

bit	int	word	long	real	acc	const	var
•							
изменение типа на:					ничто		

Перескок условно на метку, если значение аккумулятора устанавливается на FALSE (ошибочное). Используйте команды JMPC и JMPNC для ветвления после сравнения.

команда:	CAL subroutine

bit	int	word	long	real	acc	const	var
изменение типа на:					ничто		

Обязательно вызывается подпрограмма. После окончания выполнения подпрограммы, программа продолжает выполняться после команды CAL.

команда:	CALC subroutine

bit	int	word	long	real	acc	const	var
•							
изменение типа на:					ничто		

Вызов подпрограммы, если значение аккумулятора TRUE (истинное).

команда:	CALNC subroutine

bit	int	word	long	real	acc	const	var
•							
изменение типа на:					ничто		

Вызов подпрограммы, если значение аккумулятора FALSE (ошибочное). Используйте команды CALC и CALNC для условного вызова подпрограмм после сравнения.

Структурированный текст

Структурированный текст является языком высокого уровня с синтаксисом в чем-то похожим на Pascal, но он специально создан для приложений связанных с управлением промышленными процессами.

Операции назначения

Операции назначения используются для сохранения значения в переменной. Операция назначения имеет следующий общий формат:

```
variable :=  expression;
```

Приписанное значение должно быть равно или ниже типа данных, чем переменной.

Выражения

Выражения используются для вычисления значений полученных из других переменных и констант. Выражение всегда дает значение для специального типа данных. Выражение может включать одну или больше констант, переменных, операторов и функций. Используя выражения, CyBro может выполнять сложные арифметические вычисления, включающие вставленные скобки и/или различные типы данных.

Примеры:

```
y position :=  5;
down timer . pt : 15000;
circle area : = r*r*3.14;
case counter : case counter+1;
volts :=amps*ohms;
start := (oil press and steam and pump) and not emergency stop;
valid_value := (value = 0) or ((value > 10) and (value <= 60));
```

Операторы

CyBro поддерживает ряд арифметических и логических операторов, перечисленных в следующей таблице:

operator	alias	unary	binary	bit	word	int	long	real	result type
+			•			•	•	•	монотонный
-		•	•			•	•	•	монотонный
*			•			•	•	•	монотонный
/			•			•	•	•	монотонный
mod	%		•			•	•		монотонный
not	!	•		•	•				монотонный
and	&		•	•	•				монотонный
or	|		•	•	•				монотонный
xor			•	•	•				монотонный
=	==		•	•	•	•	•	•	бит
< >	!=		•	•	•	•	•	•	бит
<			•			•	•	•	бит
<=			•			•	•	•	бит
>			•			•	•	•	бит
>=			•			•	•	•	бит
:=			•	•	•	•	•	•	монотонный

Операторы сортируются по старшинству. Некоторые операторы имеют имена, которые могут использоваться вместо стандартной мнемоники.

Оценка выражения

Выражения оцениваются в специальном порядке в зависимости от старшинства операторов и других подвыражений. Выражения в скобках имеют самое высокое старшинство. Операторы самого высокого старшинства оцениваются первыми, с последующим более низким старшинством операторов до самых низких. Операторы одного старшинства оцениваются слева направо.

Рассмотрим следующий пример:

```
Speed1 := 50 . 0;
Speed2 := 60 . 0;
Press  := 30 . 0;
Rate   := Speed1/10 + Speed2/20 – (Press + 24) / 9;
```

Порядок оценки:

```
Speed1/10    =  5
Speed2/20    =  3
Press+24     =  54
54/9         =  6
8-6          =  2
Rate         =  2
```

Для изменения порядка оценки добавьте скобки:

```
Rate := Speed1/10 + Speed2/ (20 – (Press + 24) / 9);
```

Выражение (20-(Press+24)/9) имеет более высокое старшинство и будет оцениваться перед значением, использованным как делитель для Speed2.

Значение для Rate в этом случае будет:

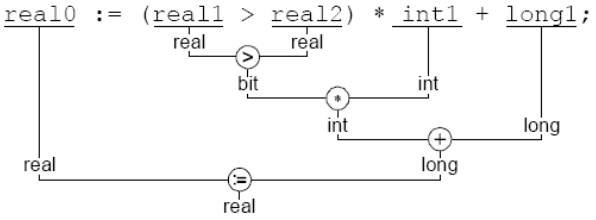
```
       =  5 + 60 / (20 – 6)
       =  5 + 60 / 14
Rate   =  9 . 286
```

Преобразования типа данных

Преобразования типа выполняются автоматически, но действительны только преобразования типа от более низкого типа к более высокому типу.

bit → integer → word → longint → real

В следующем примере будет выполняться ряд преобразований.



Если оба аргумента являются целыми числами, результат также целое число, даже если задавалась реальная переменная.

```
i    := 25
r    := i / 10;     // result  is  r=2
```

Для получения правильного результата с плавающей точкой, по крайней мере, один из операторов должен быть реальным.

Для правильности предыдущего примера константа 10 может быть записана как 10.0:

```
i    := 25
r    := i / 10.0;       // result   is  r=2 . 5
```

Тот же самый результат получается приведение одного оператора к реальности:

```
i    := 25
r    := real (i) / 10;      // result   is  r=2 . 5
```

Многострочные выражения

Есть возможность записи многострочного выражения, но каждая строка должна заканчиваться оператором. Следующий пример показывает, как выражение может разделяться на несколько строк.

```
heater   on := (heater temperature < 600)   and
                (((mode   =   MANUAL   and start pressed)   or
                  ((mode =   AUTO    and heater on request))    and
                not emergency_stop;
```

Условные операторы

Условные операторы обеспечивают ограниченное выполнение блоков операторов, в зависимости от истинности или когда нет соответствующих специальных условий.

if…then…else (если…затем…еще)

Блок операторов может оцениваться и использоваться в зависимости от значения возвращенного логического выражения используя конструкцию if…then (если…затем).

Выражение имеет общую форму:

```
if    <boolean expression> then
      <statements>;
end_if
```

Логическим выражением может быть любое выражение, которое дает логический результат TRUE (истинное) или FALSE (ошибочное), например, состояние однобитового переменного или комплексное выражение включает цифровые переменные.

Альтернативные операторы могут работать с использованием конструкции if…then…else (если…затем…еще) в форме:

```
if    <boolean expression> then
      <statements>;
elsif <boolean expression> then
      <statements>;
else
      <statements>;
end_if
```

Примеры успешного исполнения:

```
if    collision  then
      speed :=0;
      brakes :=ON;
end  if;
if    (gate=closed)   and   (pump-on)   and   (temp>200)  then
      control state :=active;
else
      control state :=hold;
      pump speed :=10;
end_if
```

Конструкции if…then (если…затем) и if…then…else (если…затем…еще) могут формироваться в пределах других условных операторов для создания более сложных условных операторов.

```
if flow rate > 230 then
    if flame size > 4 then
      fuel :=4000;
    else;
      fuel :=2000;
    end if;
else
    fuel :=1000;
end_if
```

Добавочные операторы могут условно исполняться в пределах if…then (если…затем) используя конструкцию elsif, которая имеет общую форму:

```
if  <boolean expression> then
    <statements>
elsif  <boolean expression> then
    <statements>
else
    <statements>
end_if
```

Любое число дополнительных сегментов elsif может прибавляться к конструкции if…then (если…затем).

```
if  a>b then
    d :=1;
elsif     a=b+2     then
    d :=2;
elsif     a=b-3     then
    d :=4;
else
    d :=3;
end_if
```

case…do (выбор…из)

В условном смысле оператор выбора может обеспечить четкую альтернативу для глубоко вложенных множеств. Он состоит из выражения выбора и списка операторных блоков, каждый предваряется одним возможным значение выражения. Значение выражения выбора должно относиться к перечисляемому типу (логическое, целое число или длинное целое число) и может быть следствием другого сложного выражения.

Набор операторов, который имеет постоянное значение, которое согласует значение выполняемого выражения. Если согласие не найдено, будет выполняться оператор предшествующий else (еще).

Конструкция выбора имеет следующую форму:

```
case <expression>    of
    <value1>:      <statements>;
    <value2>:      <statements>;
    <value3>:      <statements>;
else
    <statements>;
end_case;
```

Пример:

```
case material type of
    1 :   speed :=5;
    2 :   speed :=20;
    3 :   speed :=25;
          fan :=ON
    4 :   speed :=30;
          fan :=ON;
    5 :   speed :=50;
          fan :=ON;
          water :=ON;
else
    speed :=0;
end  case;

case alarm bit of
    TRUE :     speed :=0;
    FALSE :    speed :=MAX SPEED;
end_case
```

Итерационные операторы

Итерационные операторы используются для ситуаций, где необходимо повторять одну или больше операцию за некоторое время, зависящее от состояния специальной переменной или условия.

Итерационные операторы должны использоваться осторожно для того, чтобы избежать бесконечного зацикливания, которое может быть причиной ошибки переполнения скана.

for…do (для…исполнить)

Конструкция for…do (для…исполнить) позволяет набору операторов повторяться в зависимости от значения итерационной переменной. Это переменная целого числа или переменная длинного целого числа, которая используется для счета выполнения действия операторов. Итерационная переменная приращивается на 1 до конца цикла for…do (для…исполнить).

Эта конструкция принимает обычную форму:

```
for <var>   :=<expression>      to  <expression>    do
    <statements>;
end_for;
```

Перед проведением итерации переменную тестируют, чтобы получить окончательное значение. После прекращения действия конструкции for…do (для…исполнить) значение итерации будет содержать окончательное значение выражения.

Операторы в пределах цикла for…do (для…исполнить) не должны содержать команд fp или fn.

Пример:

```
for i :=0     to  19 do
    channel [i] :=TRUE;
end_for;
for t :=lo    value-1    to  hi  value*2   do
    tank number :=t
    state [t] :=t/2;
end_for;
```

while…do (тогда как…исполнить)

Конструкция while…do (тогда как…исполнить) позволяет работать одному или более операторов, тогда как специальное логическое выражение остается истинным. Перед действием операторов логическое выражение тестируется. Если оно ошибочное, операторы внутри конструкции while…do (тогда как…исполнить) не работают.

Эта конструкция принимает обычную форму:

```
while <expression> do
    <statements>;
end_while;
```

Операторы в пределах цикла while…do (тогда как…исполнить) не должны содержать команд fp или fn.

Пример:

```
while value < (max_value-10)    do
    value :=value+position;
end_while;
```

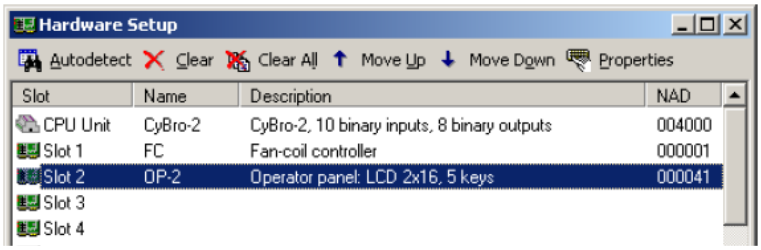
Цикл while…do (тогда как…исполнить) редко используется в обычной программе контроллера (PLC).

Панель управления

Общее описание

Панель управления (ПУ) является опциональным внешним устройством, присоединяемым к CyBro-2 с помощью шины IEX-2. ПУ имеет ЖК дисплей и несколько кнопок читаемых из программы контроллера (PLC).

Для правильной работы ПУ должна определяться в диалоговом окне Hardware Setup (настройка аппаратных средств). Настройка аппаратных средств сохраняется вместе с проектом.



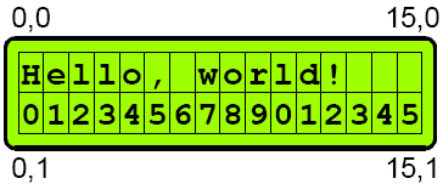
Для программирования панели управления доступны следующие инструменты:

Функции печати	Функции структурированного текста определенные в программе контроллера (PLC). Используется для показа строк и значений.
Кнопки ПУ	Битовые переменные читаются из программы (PLC). Представляют кнопки панели управления.
Маски	Визуальный инструмент для программирования панели управления, используемый для ввода параметров. Возможен ввод целочисленных значений и значений, представляемых строками. Параметры могут быть организованы иерархически.

Функции печати

Функции печати являются функциями структурированного текста, используемыми для показа текстовых сообщений и значения.

Первым параметром является номер слота, где появляется дисплей в настройке аппаратных средств. Два следующих параметра всех функций являются координатами x и y. Они используются для установки положения дисплея. Печать находится в верхнем левом углу.



Не допускается печать вне видимого поля, поскольку это может дать неожиданные результаты.

Функции печати:

```
dclr (slot : int);
```

Дисплей чист.

Очищен весь дисплей. Не требуется никаких параметров.

```
dprnc   (slot : int,    x : int,    y : int,    c : char);
```

Дисплей печатает символы ASCII.

Печать одного символа на указанных координатах. Символ может вводиться прямо (‘A’), как ASCII константа (65) или используя целочисленную переменную, которая представляет ASCII значение. Допускаются значения от 0 до 255.

```
dprns   (slot : int,    x : int,    y : int,    str : string);
```

Дисплей печати строки.

Печать строки символов на указанных координатах. Строка представляет собой набор символов заключенных в одинарные кавычки. Она может содержать любые символы из расширенного набора ASCII (символьные коды от 32 до 255).

```
dprnb   (slot : int,    x : int,    y : int,    c0 : char,    c1 : char,    value : bit);
```

Дисплей печати двоичного значения.

Печать первого или второго ASCII символа на указанных координатах, в зависимости от битного значения. Если значение ошибочное, печатается первый символ, в других случаях второй.

```
dprni   (slot : int,    x : int,    y : int,    w : int,        zb : bit,       value : int);
```

Дисплей печати целочисленного значения.

Печать целочисленного значения на указанных координатах. Параметр w определяет ширину печати. Например, если w соответствует 4, допустимый диапазон печати от -999 до 9999. Параметр zb является двоичным и устанавливается для бланкирования нуля. Если zb устанавливается на 1, передние нули не печатаются.

```
dprnl   (slot : int,    x : int,    y : int,    w : int,        zb : bit,       value : long);
```

Дисплей печати длинных значений.

Печать длинных значений на указанных координатах. Параметр w определяет ширину печати. Например, если w соответствует 6, допустимый диапазон печати от -99999 до 999999. Параметр zb является двоичным и устанавливается для бланкирования нуля. Если zb устанавливается на 1, передние нули не печатаются.

```
dprnr   (slot : int,    x : int,    y : int,    w : int,        dec : int,      value : real);
```

Дисплей печати реальных значений.

Печать реальных значений в указанных координатах. Параметр w определяет ширину печати и параметр dec определяет число печатаемых десятичных знаков. Например, если w соответствует 6 и dec соответствует 2, допустимый диапазон печати от -99.99 до 999.99. Бланкирования нуля всегда включено.

Каждый функциональный параметр (кроме строки для dprns) может быть константой, переменной или выражением. Он может использоваться для создания анимированного дисплея, как в следующем примере:

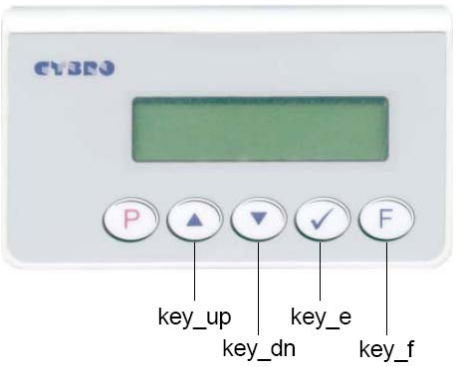
```
dclr (1);
dprns (1, 0, 0, ‘Moving…’);
dprns (1, x, 1, ‘o’);
x : = (x+fp (clock_100ms)) %16;
```

CyBro-2 может оснащаться несколькими панелями управления.

Кнопки ПУ

Кнопки ПУ доступны из программы контроллера (PLC) как входные битовые переменные.

Доступные кнопки:



Кнопка P является резервной и не доступна из программы контроллера (PLC). Ее функция должна запрашивать систему масок. Когда маска активна, кнопки up (вверх), dn (вниз) и E также резервируются, так их значения в программе котроллера (PLC) нулевые.

Кнопка не имеет функции автоповторения. Переменная кнопки является истинной все время, пока кнопка нажимается. После отпускания кнопки, переменная кнопки становится ошибочной.

Любые две (или больше) кнопок могут нажиматься одновременно. Это может использоваться для инициации некоторых вредных или редко используемых операций. В следующем примере одновременное нажатие up (вверх) и dn (вниз) будет переустанавливать переменную product_count.

```
if fp (key up and key dn) then
  product count :=0;
end_if
```

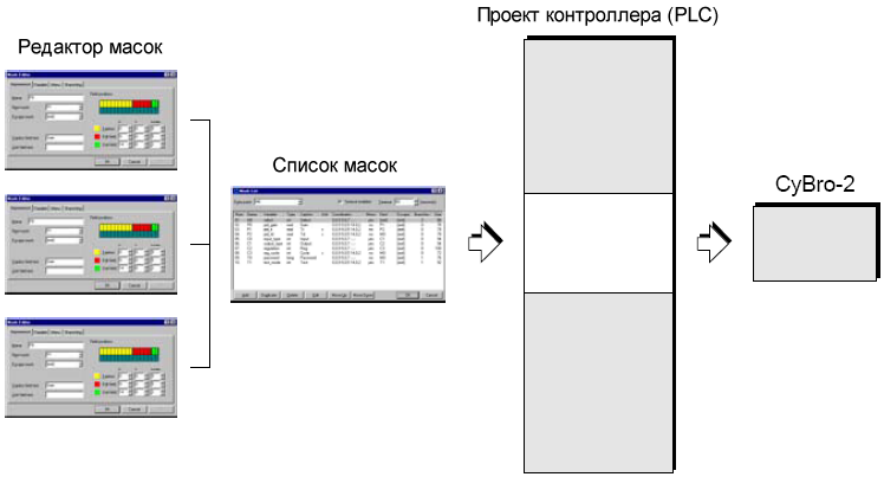
Переменные кнопки автоматически выделяются при определении ПУ в Настройке аппаратных средств.

Маски

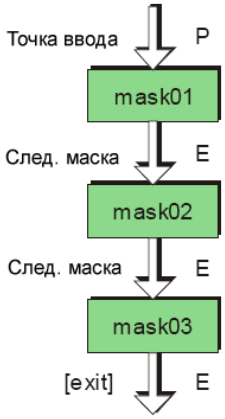
Система масок является легким в пользовании визуальным инструментом для программирования терминала оператора.

Маска является контейнером для переменной, которая будет редактироваться. Маски передаются в CyBro вместе с кодом контроллера (PLC).

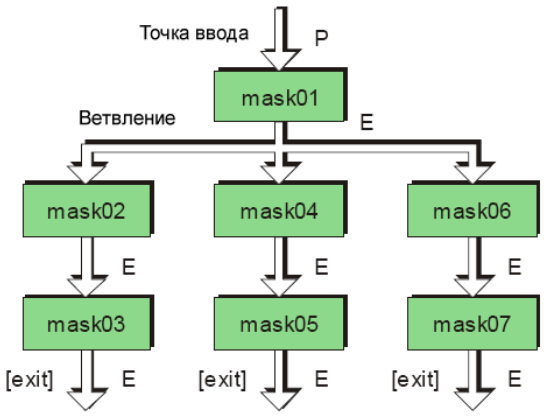
Пользователь создает новую маску или редактирует существующую с помощью Mask Editor (редактор маски). Созданные маски записываются в Mask List (список масок). Маски являются составной частью проекта контроллера (PLC), они сохраняются на диске и передаются в CyBro.



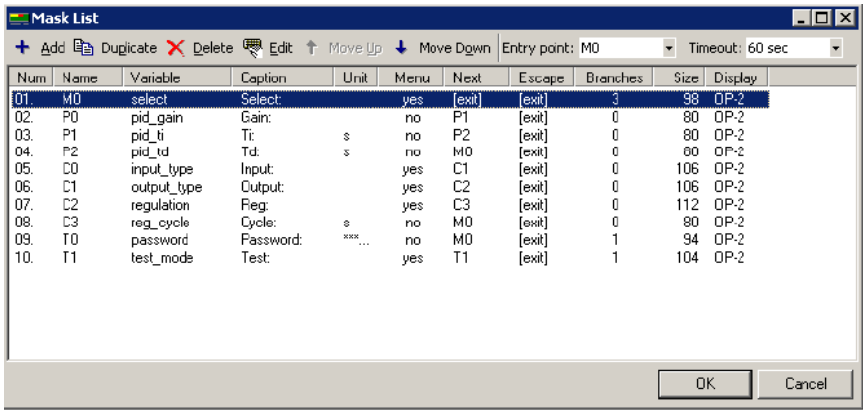
Когда пользователь нажимает P на терминале оператора, CyBro посылает первую маску в ПУ. Нажатие E дает передвижение на следующую маску.



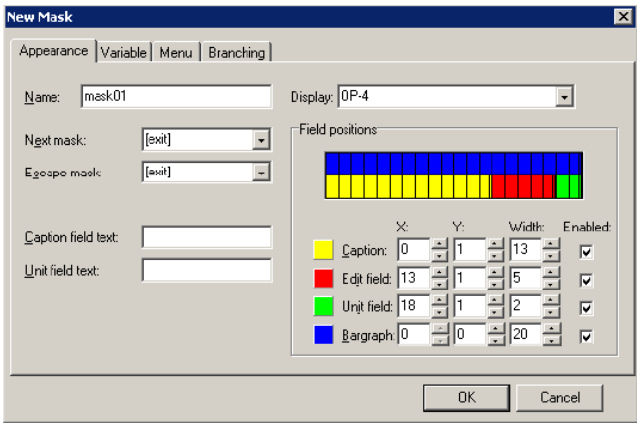
При использовании ветвящихся функций маски могут организовываться иерархически:



Для запуска работы с масками нажмите кнопку Masks (маски) на стандартной панели инструментальных средств или нажмите кнопку F7. Появится диалоговое окно Mask List (список масок).



Для создания новой маски кликните на Add (добавить) или нажмите кнопку Insert (вставка). Появится диалоговое окно Mask Editor (редактор масок).



Name (имя) является уникальным идентификатором строки, которое идентифицирует специальную маску.

Next mask (следующая маска) определяет маску, которая становится активной после нажатия кнопки E.

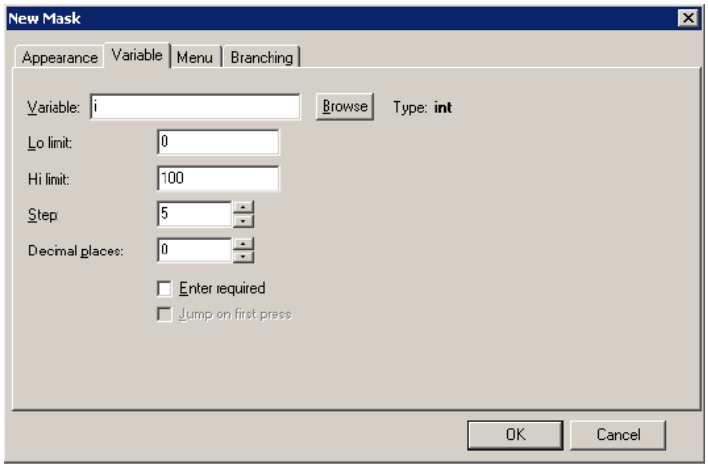
Escape mask (сброс маски) определяет маску, которая становится активной после нажатия кнопки P. Обычно, эта кнопка используется для выхода из маски.

Caption field (поле заголовка) является короткой строкой, которая будет появляться на дисплее для идентификации текущей редактируемой переменной. Положение заголовка определяется желтым прямоугольником. Для движения заголовка перетащите прямоугольник в желаемое положение. Для изменения размера заголовка потяните за правый край прямоугольника.

Edit field (поле редактирования) является зоной дисплея, в которой показывается значение отредактированной переменной. Она представляется красным прямоугольником. Поле редактирования должно иметь достаточно места для редактируемой переменной в желаемом диапазоне. Для перемещения и изменения размера поля произведите действия как для заголовка.

Unit field (поле единиц) является короткой строкой, подобной заголовку. Поле единиц представляется зеленым прямоугольником и оно обычно используется для показа единиц измерений.

Bargraph (гистограмма) является полуграфической горизонтальной строкой состояния. Доступны несколько различных стилей. Для использования гистограммы должны определяться как нижний, так и верхний пределы.



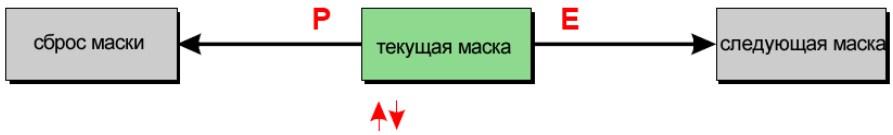
Lo limit (нижний предел) и Hi limit (верхний предел) определяет разрешенный диапазон.

Step (шаг) определяет значение, для которой при одном нажатии кнопки будет изменяться переменная.

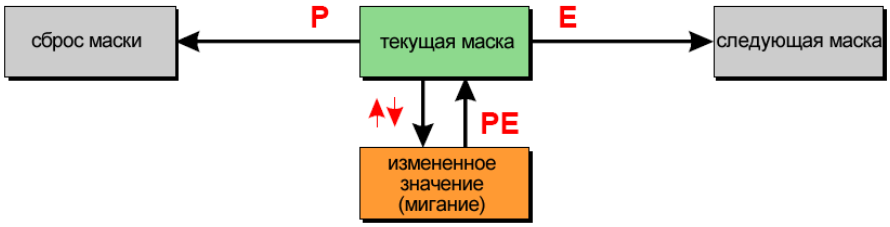
Decimal places (позиция десятичной точки) может использоваться для реальных, а также для целых и длинных целых переменных. Это относится только к показу дробей (например, для позиции десятичной точки = 1, значение 254 показывается как 25.4).

Enter required (обязательный ввод) и Jump on first press (переход на первое нажатие) определяют метод работы с навигационными кнопками (P, E). Доступны три комбинации:

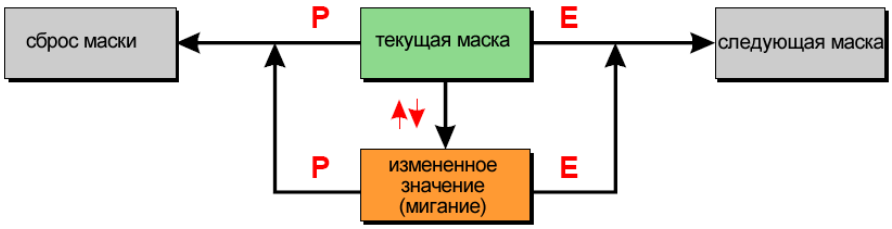
Обязательный ввод: нет



Обязательный ввод: да
Переход на первое нажатие: нет

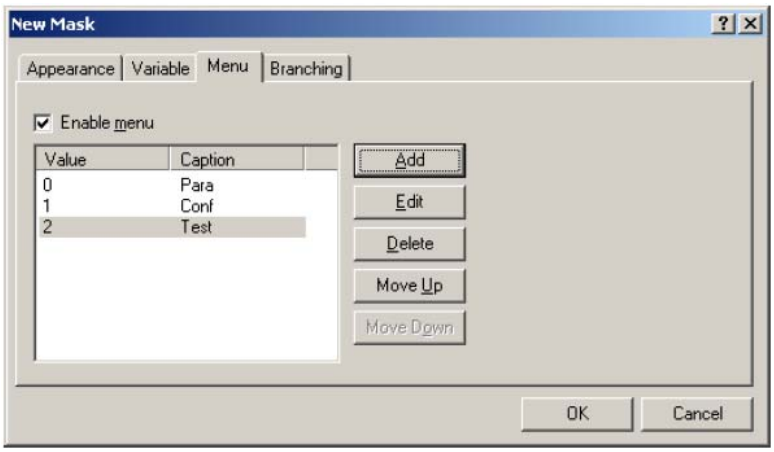


Обязательный ввод: да
Переход на первое нажатие: да



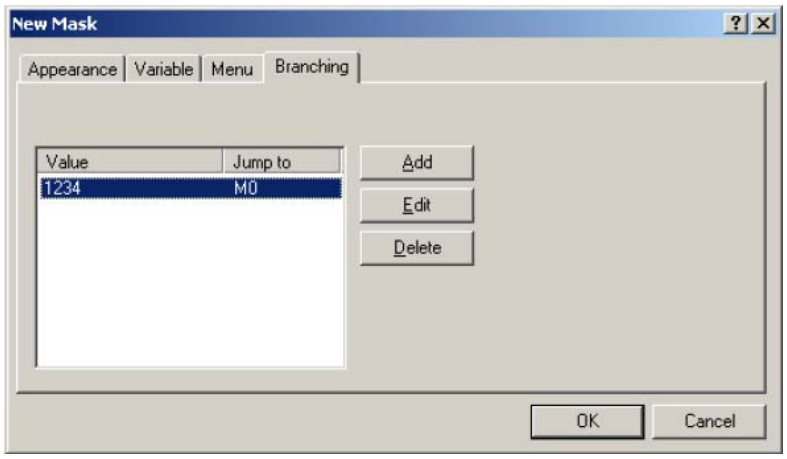
Если обязательный ввод ошибочный, измененное значение будет посылаться в CyBro немедленно после нажатие кнопки up (вверх) или dn (вниз). Если обязательный ввод истинный, измененное значение будет посылаться в CyBro только, когда нажимается кнопка E. Для индикации того, что изменение не подтверждено, измененное значение будет мигать.

Некоторые переменные могут вводится, как меню, вернее, как цифровое значение. Для задания ввода меню запустите Mask Editor (редактор маски), кликните таблицу Menu (меню) и Add (добавление) так как необходимо иметь много позиций.



При выполнении программы CyBro дисплей будет показывать позиции имени и переменная product_type будет принимать значение 0, 1 или 2.

Ветвящиеся таблицы обеспечивают ветвление в различных масках в соответствии с введенным значением. Это можно использовать для организации параметров в различные наборы параметров, но также для параметров защищенных паролем.



Активная маска управляет всеми кнопками панели за исключением кнопки F, поэтому в то же самое время невозможно использовать их из программы CyBro. Поля маски показываются «над» дисплеем пользователя. После свертки маски содержимое дисплея возвращается в прежнее состояние.

Если маска слишком большая для работы с панелью управления, она не будет активироваться и она будет работать как пустая маска. Размер маски показывается в диалоговом окне Mask List (список масок). Доступная память для маски панели управления показывается в диалоговом окне Hardware Setup (настройка аппаратных средств). Для уменьшения размера маски уменьшите число вводов в меню или уменьшите ширину редактируемого поля. Уменьшение ширины поля заголовка и единиц измерений также может сохранить несколько байтов.

Невозможно активировать одновременно дополнительные маски.

Оперирование масками из программы контроллера (PLC)

Программа CyBro может получить текущую активную маску считыванием переменной current_mask. Когда current_mask является нулем, нет активных масок.

Программа может принудительно выполняться для определенной маски записью переменной next_mask. После пересылки маски, next_mask будет настраиваться на -1 и соответствующим образом будет изменяться current_mask.

Следующий пример показывает типичную смену маски:

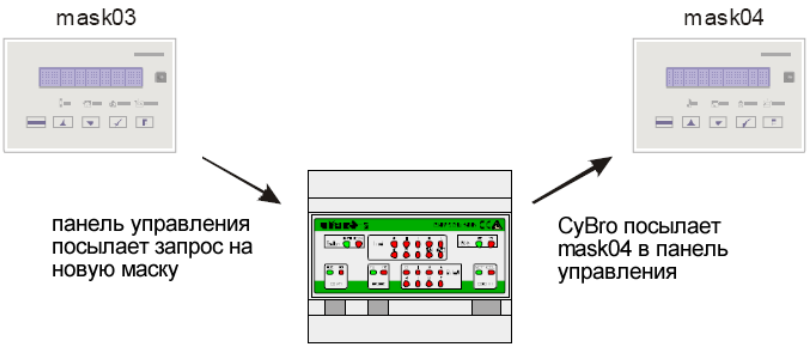
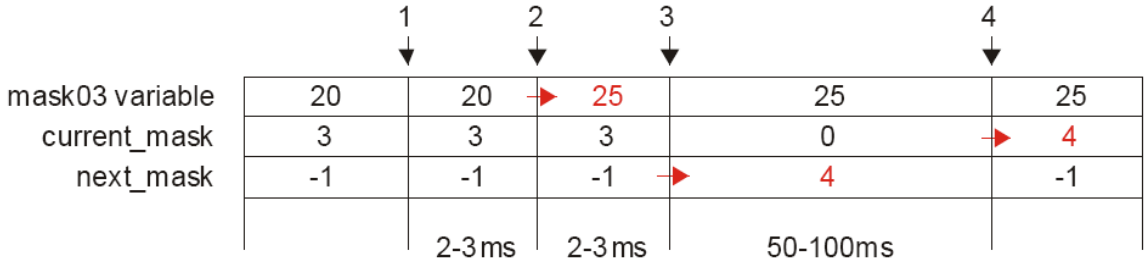


Таблица показывает примерный расчет времени и значений для смены:



Ход событий маркируется черными стрелками:

1. Пользователь нажимает Enter (ввод)
2. Новое значение посылается в CyBro
3. Требование на новую маску посылается в CyBro
4. Новая маска посылается в панель управления и активируется

Красные стрелки обозначают наиболее важные изменения значения.

Такая же смена может активизироваться следующей программой контроллера (PLC):

```
if    <condition> then
   op00    next   mask :=4;
end_if
```

Короткий пробел в значении current_mask получается из ограниченного времени отклика сети. Для проверки того, что нет активной маски, программа также должна проверять значение next_mask. Следующий пример будет правильно настраивать начальную маску панели:

```
if    op00  current    mask=0  and    op00  next    mask=-1 then
   op00    next   mask :=10;
end_if
```

Обе переменных маски управления также могут быть доступны через шину A-bus.

Порт Com2

Общее описание

Порт Com2 является многоцелевым последовательным портом RS232.

Доступны следующие режимы работы:

- Второй A-bus порт
- Коммуникационный порт общего назначения
- Порт недоступен, активен счетчик HSC

Для переключения режимов Com2 используйте следующие команды контроллера (PLC):

A-bus режим:

```
com_init (0, 0, 0, 0);
```

Режим общего назначения:

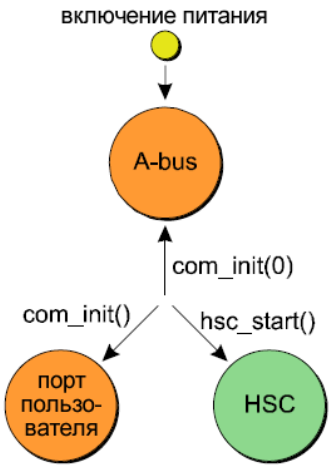
```
com_init (baud_rate,  data_bits,  parity,  stop_bits);
```

Режим HSC:

```
hsc_start ( ) ;
```

Другие команды (rx_stop(), tx_stop(), hsc_stop()) могут использоваться только, когда активируется соответствующий режим.

Приведенная диаграмма суммирует допустимые режимы и перемены:



Режим A-bus включается по умолчанию (активируется при включении питания и смены остановка-работа).

Нельзя использовать два режима одновременно, но есть возможность переключения режимов во время выполнения программы контроллера (PLC).

Например, если контроллер (PLC) присоединяется к модему с использованием порта Com2, режим общего назначения для выполнения вызова (посылка AT команд), затем режим A-bus может использоваться для передачи данных (присоединяя к CyBro OPC сервер) и затем опять контроллер (PLC) может переключаться на режим общего назначения для присоединения к линии.

Режим A-bus

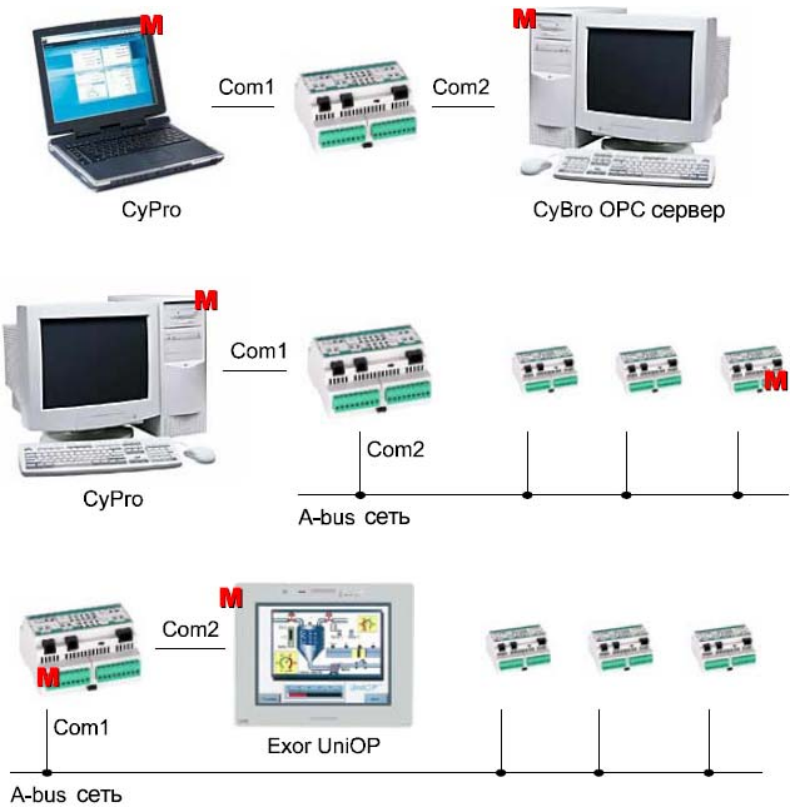
Используя режим A-bus порт Com2 может присоединяться к:

- CyPro
- CyBro OPC серверу
- Панели управления Exor UniOP
- Другим CyBro в сети

Порт Com2 поддерживает только режим подчиненного узла, он не может быть ведущим сетевым узлом. Работа ведущим узлом ограничивается портом Com1.

Порты Com1 и Com2 полностью независимы – контроллер (PLC) может одновременно посылать и принимать сообщения на оба порта.

Некоторые примеры присоединения показаны ниже. “M” стоит на ведущих сетевых узлах.



CyBro не может быть сетевым входом – сообщения не передаются от одного порта на другой.

Режим общего применения

Как к коммуникационному порту общего применения к нему могут присоединяться несколько последовательных устройств: сенсоры, шкалы, модемы, радио соединения, принтеры и другие.

Коммуникационный протокол определяется программой контроллера (PLC). Хотя поддерживается двоичный формат сообщения, коммуникационные функции наиболее хорошо подходят для посылки, получения и анализа ASCII сообщений. Поддерживается как ведущая, так и ведомая работа. Порт Com2 является полностью дуплексным, так что данные могут проходить одновременно в обоих направлениях.

Инициализация

Перед любой другой команде по коммуникации порт Com2 должен инициализироваться. Для инициализации порта используйте функцию:

```
com_init (baud_rate : long,data_bits : int,   parity : int,   stop_bits : int)   : bit ;
```

Параметр baud_rate определяет скорость коммуникации, выраженную как число битов за секунду (бит/сек). Стандартные скорость передачи:

1200
2400
4800
9600
19200
38400

Другие доступные скорости передачи могут вычисляться следующим образом:

baud_rate = $\frac{384000}{N}$, где N является целым числом между 1 и 65536.

Пример:

Требуемая скорость передачи 300 бит/сек. Для проверки доступности этой скорости разделите 384000 на 300. Результатом будет 1280 без дроби, так что порт может инициализироваться со скоростью точно 300 бит/сек.

Пример:

Требуемая скорость передачи 56 кбит/сек. Деление 384000 на 56000 дает 6,86, так что 56 кбит/сек недопустимо. Ближайшая возможная скорость 384000/7 = 54857, которая на 2% меньше требуемой скорости. В некоторых приложениях это может быть допустимо, но это лучше выбора другой скорости передачи.

Максимальная скорость передачи 384 кбит/сек

Параметр data_bits может быть 7 или 8.

Значение параметра parity может быть:

0 – нет четности
1 – проверка на четность
2 – контроль четности

Параметр stop_bits может быть 1 или 2.

Функция com_init() становится истинной, если инициализация успешна, ошибочная, если нет. Инициализация может нарушиться при выходе параметра инициализации из диапазона.

Общая последовательность инициализации такова:

```
if first scan then
    com open : = com init (19200, 8, 0, 1);
end_if;
```

В этом примере инициализируется порт Com2 на 19200 бит/сек, 8 битовые данные, нет четности и 1 стоповый бит. Переменная com_open может использоваться для проверки, если инициализация была успешной.

Для переключения назад на режим A-bus используется специальный синтаксис com_init() со всеми параметрами, установленными на нуль:

```
com_init (0, 0, 0, 0) ;
```

После выполнения команды все другие коммуникационные функции становятся неактивными, порт инициализируется на параметры A-bus и он теперь готов для получения команд A-bus.

Подготовка сообщения о передаче

Перед передачей программа контроллера (PLC) создает сообщение о полном выходе. Для создания сообщения используют показанные функции dprnc(), dprns(), dprnb(), dprni(), dprnl() и dprnr().

Для печати в буфер передачи используйте 0 в качестве номера слота, т.е. в слоте 0 появляется буфер передачи как панель управления.

Координата X является положением буфера передачи. Координата Y всегда должна быть нулевой. Максимальная длина сообщения 1024 байт.

Например, для записи сообщения “Hello!” вводят:

```
dprns (0, 0, 0, ‘Hello!’) ;
```

Длина сообщения 6 символов.

Специальные символы могут вводиться как двух-символьная комбинация. Первым символом будет обратный слэш (‘\’) и второй будет один из следующих:

комбинация	ASCII код	Шестнадцатеричный код
\n	CR LF	0D 0A
\r	CR	0D
\t	TAB	09
\\	\	5C
\nn	любой	nn

Последняя трех-символьная комбинация может использоваться для ввода шестнадцатеричного кода любого символа ASCII. Например, ‘\41’ эквивалента букве ‘A’.

Например, для создания сообщения “Hello!” следует сделать возврат каретки и перевод строки, вводят:

```
dprns (0, 0, 0, ‘Hello!\n’) ;
```

Длина сообщения 8 символов.

Для создания сообщения выполненного из ключевых слов и цифровых значений используют:

```
dprns (0, 0, 0, '>LEVEL=xxx.xx TEMPERATURE=xx.x    ERROR=xx\n') ;
dprnr (0, 7, 0, 6, 2, water_level);
dprnr (0, 26, 0, 4, 2, water_temperature);
dprni (0, 37, 0, 2, no, error_code);
```

Заметим, что x-ы в команде dprns будут переписываться последующими командами.

Функция dprnc() могут использоваться для ввода любой одно-байтовой двоичной величины.

Например, для создания 4-байтового сообщения содержащего буквы 'ABC' выводимые из ESC символа (ASCII код 27), вводят:

```
dprnc (0, 0, 0, 'A') ;
dprnc (0, 1, 0, 'B');
dprnc (0, 2, 0, 'C');
dprnc (0, 3, 0, 27);
```

Для создания 2-байтового двоичного сообщения содержащего целую величину (наиболее значительный байт первый), вводят:

```
dprnc (0, 0, 0, value/256) ;
dprnc (0, 1, 0, value%256);
```

Используя функции dprns() и dprnc() может быть создано любое двоичное сообщение.

Передача

Для запуска передачи используют функцию tx_start():

```
tx_start (char_num : int) ;
```

Параметр char_num определяет количество символов для передачи. Передача всегда начинается с начала буфера. Нет возможности для начала передачи с любого возможного положения.

```
tx_active () : bit;
```

Функция tx_active() возвращает на текущее состояние передачи. При окончании передачи tx_active() падает до нуля.

```
tx_count () : int;
```

Функция tx_count() определяет число символов остающихся для передачи. Если tx_count() нуль, но tx_active() еще истинное, теперь передается последний символ.

```
tx_stop () ;
```

Если еще идет передача и передача требует определенное условие для немедленной остановки, может использоваться функция tx_stop(). Теперь переданный символ будет передаваться полностью, но остальное сообщение не будет передано.

Получение

Получение полностью не зависит от передачи.

Максимальная длина переданного сообщения 1024 байта. Если получается больше чем 1024 символа, положение полученного переворачивается на начало полученного буфера. В этом случае число полученных символов настраивается на нуль.

Для запуска получения используют функцию rx_start(). Эта функция также определяет критерий для остановки получения:

```
rx_start (beg_ch : char,  end_ch : char,   len : int,  msg_tout : int,  char_tout : int) ;
```

Параметр begch указывает первый символ полученного сообщения. После инициации получения, любой другой символ будет отвергаться до тех пор, пока получается begch. Для получения сообщения без указания первого символа, внесите нуль вместо begch.

Параметр endch указывает последний символ полученного сообщения. После получения endch, прием останавливается. Для получения сообщения без указания последнего символа, внесите нуль вместо endch.

Параметр len указывает длину полученного сообщения. После требуемого числа байтов прием останавливается. Для получения сообщения неопределенной длины установите len на нуль.

Параметр msg_tout указывает истечение времени для сообщения в миллисекундах. Это время, после которого получатель будет выходить из системы, если нет полученных символов. Максимальное время выхода из системы 32 секунды. Для получения без ограничения времени сообщения, настройте msg_tout на нуль.

Параметр char_tout указывает время получения между отдельными символами. Если символы посланные сопровождающим устройством посылаются непрерывно без пробелов, превышение времени для символа может устанавливаться на благоприятное низкое значение для того, чтобы улучшить время отклика получения. Превышение времени для символа всегда должно быть больше, чем время необходимое для передачи одного символа, относительно скорости передачи и битовых данных.

Пример:

Коммуникационными параметрами являются 1200 бит/сек, 8 бит и нет четности. Передача одного символа будет происходить примерно 8 мсек (стартовый бит + 8 битовых данных + стоповый бит = 10 бит; 10 бит/1200 бит/сек=8,3 мсек). Превышение времени для символа должно быть меньше 10 мсек, хотя 50…100 мсек является более соответствующим, если время отклика не критическое.

Для получения без превышения времени для символа установите char_tout на нуль.

Оба превышения времени полностью независимы друг от друга. Есть возможность установить превышение времени для символа и представить превышение времени для символа блокированным. И наоборот позволить, но это бессмысленно.

Несколько примеров будет иллюстрировать использование функции rx_start():

Непрерывное получение символов из последовательного порта:

```
rx_start (0, 0, 0, 0, 0) ;
```

Получение сообщения начинается с ‘>’ и оканчивается символом CR:

```
rx_start (‘>’, ‘\r’, 0, 0, 0) ;
```

Получение сообщения точно длиной 12 символов:

```
rx_start (0, 0, 12, 0, 0) ;
```

Бесконечное ожидание для первого символа, но получение останавливается на 250 мсек после получения сообщения:

```
rx_start (0, 0, 0, 0, 250) ;
```

Остановка получения на 100 мсек после последнего полученного символа или, если нет полученных символов за 5 секунд:

```
rx_start (0, 0, 0, 5000, 100) ;
```

Допускается любая комбинация критерия запуска и остановки. Для получения сообщения размером до 80 символов конец делается с Ctrl-Z (1A шестнадцатеричная) с 10 секундным превышением времени:

```
rx_start (0, '\1A', 80, 10000, 100) ;
```

Если данный критерий для остановки получения не соответствует запрашиваемому коммуникационному протоколу, определение конца сообщения разрешается в программе контроллера (PLC). Полученное сообщение должно анализироваться на лету и при удовлетворении условия остановки функция rx_stop() может использоваться для остановки получения:

```
rx_stop () ;
```

Функция rx_count() определяет число полученных символов. Функция может использоваться как для активирования получения, так и нет. Функция rx_start() переустанавливает это число на нуль.

```
rx_count () : int ;
```

Для проверки, если получение активно, используют функцию:

```
rx_active () : bit ;
```

Для проверки подробного статуса получения, используйте функцию:

```
rx_status () : int ;
```

Функция rx_status() определяет один из следующих кодов:

0 – получение активно
1 – выполняется функция rx_stop()
2 – конец определения символов
3 – затребованное число полученных символов
4 – окончание времени превышения

Пример:

Входящее сообщение не имеет завершающего символа и длина может различаться. Длина сообщения кодируется двоичным образом в четырех байтах. Для получения таких сообщений используют следующий код:

```
if rx_active () and     rx_count () >=4 then
    if rx_count () >=rx_bufrd (3)   then
        rx_stop () ;
    end_if ;
end_if ;
```

Анализ полученного сообщения

После получения программа контроллера (PLC) должна анализировать полученное сообщение. Подпрограмма для этого задания обычно должна:

- проверять формат сообщения и решить принимать сообщение или нет
- преобразовывать полученные данные в переменные контроллера (PLC)

Если формат сообщения определяется строго, процедура анализа может быть простой, но намного лучше иметь достаточно гибкую программу способную управлять изменением формата. Все сообщение можно отклонить с помощью негибкой процедуры анализа только из-за одного символа пробела добавленного где-нибудь.

Функции для анализа полученного сообщения:

```
rx_bufrd (position : int) : int ;
```

Возврат одного символа из данного положения приемного буфера. Символ преобразуется в целочисленное значение в диапазоне 0..255.

```
rx_strcmp (position : int,    str : string) : int ;
```

Сравнение приемного буфера с указанной строкой. Если строка соответствует, возвращаемое значение истинное, иначе оно ошибочное.

```
rx_strpos (position : int,    str : string) : int ;
```

Поиск для указанной строки в приемном буфере. Поиск начинается с данного положения. Если строка найдена, функция возвращает положение первого согласованного символа; иначе возвращаемым значением является -1.

```
rx_strtoi (position : int) : int ;
```

Возврат целочисленного значения десятичного числа, начиная с данного положения. Если символ в указанном положении является пробелом, ищется следующий символ до тех пор, пока не будет найдено число. Преобразование продолжается до тех пор, пока не встретится первый нецифровой символ.

```
rx_strtol (position : int) : long ;
```

Возврат длинного целочисленного значения десятичного числа, начиная с данного положения. Если символ в указанном положении является пробелом, ищется следующий символ до тех пор, пока не будет найдено число. Преобразование продолжается до тех пор, пока не встретится первый нецифровой символ.

```
rx_strtor (position : int) : real ;
```

Возврат реального значения десятичного числа, начиная с данного положения. Если символ в указанном положении является пробелом, ищется следующий символ до тех пор, пока не будет найдено число. Преобразование продолжается до тех пор, пока не встретится первый нецифровой символ.

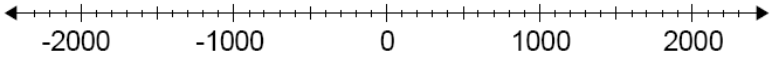
Пример:

```
if    rx  strops (0, ‘open’) <>-1    then
   main    value=1;
end  if;

if    rx  strops (0, ‘close’) <>-1    then
   main    value=0;
end  if;

position=rx    strops (0, ‘TEMP=’);
if    position<>-1    then
   set   point=rx  strtoi  (position+5);
end  if

position=rx    strops (0, ‘CYCLE=’);
if    position<>-1    then
   cycle    timer . pt=rx    strtol  (position+6);
end_if
```

Высокоскоростной счетчик

Общее описание

Высокоскоростной счетчик (ВСС) является аппаратным устройством способным подсчитывать внешние импульсы от кодировщика в обоих направлениях. Счетчик имеет 32-битный знаковый регистр.



Вход счетчика может быть однофазным, обычно называемым A, или двухфазным, обычно называемым AB. Z обозначает дополнительный вход нуля, обычно используемый для определения абсолютного положения контролируемого устройства.

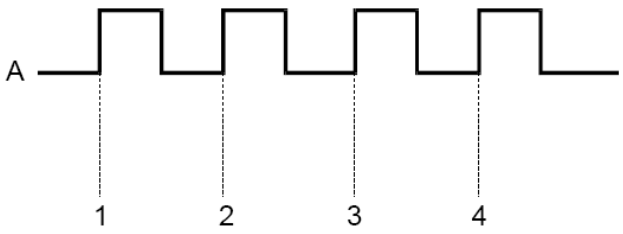
Допускается несколько режимов счета:

Нет	Высокоскоростной счетчик блокирован
Только Z	Высокоскоростной счетчик блокирован, вход нуля используется как вход по прерыванию
A/AB x1	Однофазный или двухфазный счет, одинарная точность
A/AB+Z x1	Однофазный или двухфазный счет, одинарная точность, вход нуля активен
AB x4	Двухфазный квадратично-точный счет с удалением аппаратного сбоя
AB+Z x4	Двухфазный квадратично-точный счет с удалением аппаратного сбоя, вход нуля активен

Для настройки режима счета откройте диалог Kernel Maintenance, kernel.bin, выберите желаемый режим и пошлите ядро в CyBro.

Однофазный счет

Счет в одном направлении. Используется только однофазный вход (A).

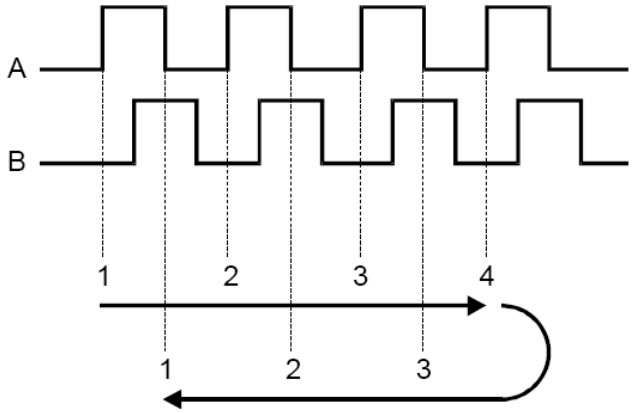


Счетчик дает приращение на переднем фронте импульса.

Направление счета может определяться состоянием B входа. Если B=0, направление счета вперед и, если B=1, направление счета назад.

Двухфазный счет, одинарная точность

Счет как вверх, так и вниз. Используются двухфазные входы (A и B).

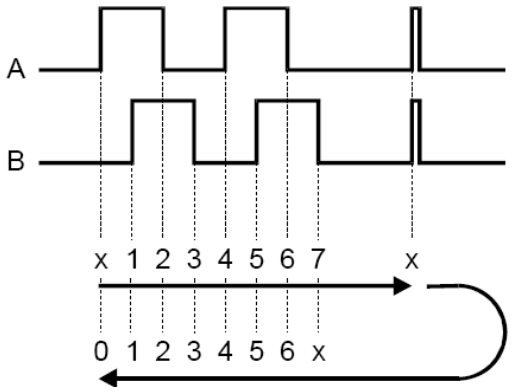


Счетчик увеличивает или уменьшает значение на переднем фронте импульса на входе A. разрешение счетчика равно разрешению кодировщика, каждый импульс кодировщика приращивает (или уменьшает) показание счетчика на 1.

Вибрация вала или электромагнитный шум может привести к ошибочному счету.

Двухфазный счет, квадратичная точность

Счет как вверх, так и вниз. Используются двухфазные входы (A и B). Счетчик приращивает или уменьшает значение на каждом фронте входов A и B. Разрешение счета соответствует четырехкратному разрешению кодировщика, так каждый импульс кодировщика приращивает (или уменьшает) показание счетчика на четыре. 500 импульсов на поворот кодировщика будет в действительности приращивать отсчет на 2000.



x – нет изменения

Режим квадратичной точности также приводит к удалению проблемы и удалению ±1 импульса, устраняющее неправильный счет по причине вызванной вибрацией вала или электромагнитного шума.

Высокоскоростной счетчик и последовательный порт Com2 не могут использоваться одновременно. Функция hsc_start() останавливает последовательную коммуникацию Com2. Для повторного запуска коммуникации должна вызываться функция com_init().

После включения питания значение высокоскоростного счетчика не определено.

Если BCC не используется, высокоскоростные входы A и B могут использоваться как стандартные двоичные входы.

Порядок присоединения и технические характеристики BCC, пожалуйста, смотрите в руководстве для аппаратных средств.

Высокоскоростной счет

По умолчанию высокоскоростной счетчик останавливается. Для запуска счетчика используют функцию hsc_start().

```
hsc_start () ;  //  start  high-speed  counter
```

Для остановки счета используют функцию hsc_stop().

```
hsc_stop () ;  //  stop  high-speed  counter
```

Для определения остановки или запуска ВСС используют функцию hsc_active(). Если запускается ВСС, функция hsc_active() становится истинной, в другом случае определяется ошибка.

```
if hsc active ()  then  //  check if  high-speed  counter  is  running
   active  drive  () ;
end_if
```

Для считывания текущего положения высокоскоростного счетчика используют функцию hsc_read(). Считывание возможно без проблем, если счетчик активен или нет. Типичным значением на счетчике является длинное целое число и оно может сохраняться в переменной того же типа.

```
current_position : =hsc_read () ;
```

Для настройки счетчика на желаемое значение используйте функцию hsc_write(). Обычно это делается только при остановке счетчика, но есть возможность записать новое значение даже, если счетчик активен и работает.

```
hsc_write (12000) ;
```

Как пример приращение показания счетчика на 500 отсчетов.

```
hsc_write (hsc_read () +500) ;
```

Высокоскоростное действие

Высокоскоростной счет обычно используется для точного контроля движения. Следующий пример показывает, как остановить двигатель, присоединенный к входу qx000, когда счетчик достигает нуля (счет вниз).

```
if   hsc   read () <=0 then
   qx000 : =0 ;
end_if ;
```

Проблема, которая здесь может появиться, состоит в непредсказуемом времени отклика. Если текущее время сканирования 5 мсек, время отклика может быть любым между 0 мсек и 10 мсек. Для скорости 1 м/сек это дает неточность в 10 миллиметров.

Для предотвращения этого высокоскоростной счетчик имеет аппаратные средства способные выполнить очень быстрые действия при достижении счетчиком нуля. Действие определяется как настройка или перенастройка одинарного двоичного выхода. Для выполнения задания как в последнем примере выполните следующее:

```
hsc_set_action (0, qx000) ;
```

Функция hsc_set_action() инициирует действие. При достижении счетчиком нуля выход qx000 будет деактивирован. Также возможно настроить действие в противоположном направлении для активации выхода. Смотрите пример:

```
hsc_set_action (1, brake_out) ;
```

При достижении счетчиком нуля будут активированы тормоза.

Действие должно инициироваться только один раз, так что типичный код активационного действия есть:

```
if    fp  (start key)   then
    hsc  write (300000) ;
    hsc  set action (0, qx000) ;
    hsc  start ();
    qx000 : =1 ;   //  start   motor
end_if ;
```

Функция hsc_reset_action() обеспечивает отмену инициированного действия. Это полезно, когда случается что-нибудь неожиданное, например, при обнаружении аварийного состояния.

```
if    alarm  condition    then
    hsc  reset action () ;
    hsc  stop () ;
    qx000 : =0 ;
    qx001 : =0 ;
end_if ;
```

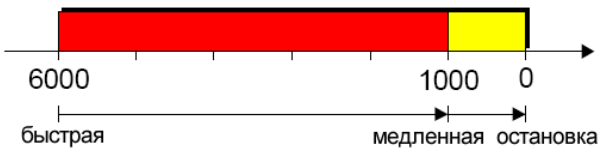
Действие выполняется только один раз, когда счетчик достигает нуля в первый раз. Когда счетчик достигает нуля опять, никаких действий не будет выполняться. Для проверки того, что действие еще происходит, используйте функцию hsc_check_action(). Если действие инициируется, функция hsc_check_action() определяется как истинная. После выполнения действия (или его отмены) функция hsc_check_action() определяется как ошибочная.

Невозможно инициировать более одного действия одновременно. Если предпринять такую попытку, то это будет причиной неправильной работы. Если есть необходимость в нескольких действиях, следующее действие может быть инициировано после выполнения первого действия. В приложениях, для которых необходимо мультискоростное управление, изменение передаточного числа также может выполняться командами if…then (если…затем), а не действия. Обычно, это позволяет достаточно плавно замедлиться с одним высокоскоростным действием, использованным для безошибочной остановки.

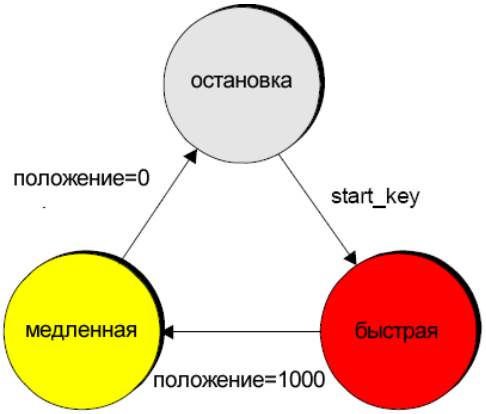
Следующий пример показывает типичное двухскоростное применение. Двухскоростной двигатель активируется двумя двоичными выходами, qx000 и qx001. Низкая скорость активируется qx000 и высокая скорость активируется qx001.

	остановка	медленная	быстрая
qx000	0	1	1
qx001	0	0	1

Пример требует движения 6000 блоков. Для первых 5000 блоков двигатель будет работать быстро, затем для следующих 1000 блоков он будет работать медленно и затем остановится. Замедление будет позволять остановиться в точном положении.



Программа состоит из трех основных стадий: остановка, быстрая и медленная. Переходы между стадиями определяются start_key и положением.



Действующий код может записываться как первым, так и вторым способом командой if…then (если…затем) или двумя последовательными действиями.

Первый способ немного короче и более легкий для понимания:

```
if     qx000=0   and     qx001=0      then
//     currently  stopped
    if     fp (start key)  then
           //   start full speed and set action to stop after 6000
           hsc write (6000) ;
           hsc set action (0, qx000) ;
           hsc start () ;
           qx000 : =1 ;
           qx001 : =1 ;
    end_if ;
elsif      qx000=1   and     qx001=1   then
//     currently  fast
    if hsc read () <=1000   then
           //   slow   down
           qx000 : =1 ;
           qx001 : =0 ;
    end   if ;
end_if ;
```

Некоторый набор функций может выполняться другим способом: двумя последовательными активационными действиями. Понятно, что код является большим или меньшим битом, но действия будут выполняться очень точно.

```
if     qx000=0   and     qx001=0      then
//     currently  stopped
    if     fp (start key)  then
           //   start full speed and set action to slow down after 5000
           hsc write (5000) ;
           hsc set action (0, qx001) ;
           hsc start () ;
           qx000 : =1 ;
           qx001 : =1 ;
    end_if ;
elsif      qx000=1   and     qx001=1   then
//     currently  slow
    if not hsc_check_action ()      then
           //   continue   slow? Increment counter by 1000 and stop on zero
           hsc_write (hsc_read () +1000) ;
           hsc_set_action (0, qx000) ;
    end_if ;
end_if ;
```

Переустановка нуля

BCC (HSC) CyBro типов A/AB+Z x1 и AB+Z x4 имеют дополнительный высокоскоростной выход, используемый для точного определения абсолютного положения.

Обычная функция входа нуля должна переустанавливать высокоскоростной счетчик. По умолчанию эта функция блокирована и может разблокироваться функцией hsc_enable_zero(). Если произошло разблокирование, каждый переход неактивность-активность на входе нуля будет переустанавливать высокоскоростной счетчик на нуль.

Для блокировки переустановки нуля используйте функцию hsc_disable_zero(). Для проверки того, что переустановка нуля разблокирована, используйте функцию hsc_check_zero(). Если переустановка разблокирована, функция hsc_check_zero() будет определяться как истинная, иначе она будет определяться как ошибочная.

Переустановка нуля не является одноразовой функцией, так что однажды активированная переустановка активирует до тех пор, пока она недвусмысленно деактивируется функцией hsc_disable_zero().

Если переустановка нуля разблокирована и действие активируется, действие будет выполняться немедленно после детектирование перехода.

Выявление нуля

Второй способ использования входа нуля обеспечивается двумя дополнительными функциями hsc_detect_zero() и hsc_read_zero().

Функция hsc_detect_zero() показывает переход на вход нуля. Если был определен переход неактивность-активность на входе нуля, функция hsc_detect_zero() будет определяться как истинная, но только первый раз. Следующие последовательные вызовы будут квалифицироваться как ошибочные до тех пор, пока не будет выявлен следующий переход на нуль.

Это позволяет сделать простой подсчет переходов:

```
zero_counter : =zero_counter  +   hsc_detect_zero () ;
```

При определении перехода значение высокоскоростного счетчика также записывается в регистр выявления нуля, считываемого функцией hsc_read_zero(). Регистр выявления нуля также длиной 32 бита.

Функции выявления нуля могут использоваться в A/AB +Z x1 и AB+ Z x4 режимах.

Часы реального времени

Общее описание

Часы реального времени (ЧРВ) являются аппаратным часовым/календарным устройством. Они работают даже, когда источник питания выключен. Технические характеристики относительно точности и времени сохранения данных, пожалуйста, смотрите руководство для аппаратных средств.

ЧРВ могут синхронизироваться по часам ПК с помощью каждой временной программы переданной в контроллер (PLC). Для разблокировки или блокировки этой опции вычистите окно Tools/Environment Options/Communication/Synchronize RTC to PC Clock. ЧРВ также могут настраиваться из программы контроллера (PLC) с помощью переменных Вх/Вых ЧРВ.

Переменные Вх/Вых

Для считывания или настройки времени ЧРВ используйте:

```
rtc   hour : int ;
rtc   min : int ;
rtc_sec : int ;
```

Диапазоны значений, в которых изменяются переменные:

Часы	0..23
Минуты	0..59
Секунды	0..59

Для считывания или настройки даты ЧРВ используйте:

```
rtc   year : int ;
rtc   month : int ;
rtc_date : int ;
```

Значения находятся в диапазоне:

Год	2000..2099
Месяц	1..12
День	1..31

Для считывания или настройки дня недели ЧРВ используйте:

```
rtc_weekday : int ;
```

0 – Воскресенье
1 – Понедельник
2 – Вторник
3 – Среда
4 – Четверг
5 – Пятница
6 – Суббота

Настройка часов реального времени, запись нового времени/даты в переменные Вх/Вых и настройка записи требует признак:

```
rtc_write_req : 1 ;
```

Примером того, что показом даты/времени на панели управления и разрешением пользователю настроить их, является RtcClock.cyp. Программа размещается в директории Project\Examples.

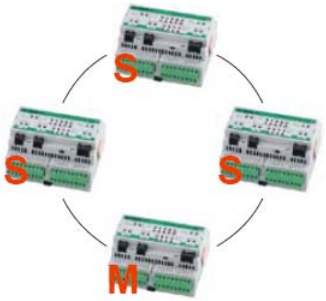
Объединение в сеть

Общее описание

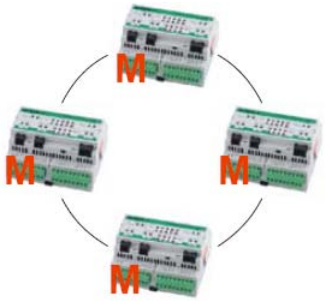
Блоки CyBro мгут соединяться в частную сеть называемую A-bus. Сеть делается для программирования и мониторинга, а также обмена данными между устройствами.

В зависимости от аппаратных средств определяются два сетевых протокола:

- протокол ведущий-ведомый (master-slave) (Com1/Com2, последовательная сеть)
- мультиведущий протокол (ETH, сетевой протокол TCP/IP)



Последовательная сеть



Сеть TCP/IP

Даже, несмотря на одинаковый формат сообщения, сеть TCP/IP обеспечивает улучшенную функциональность.

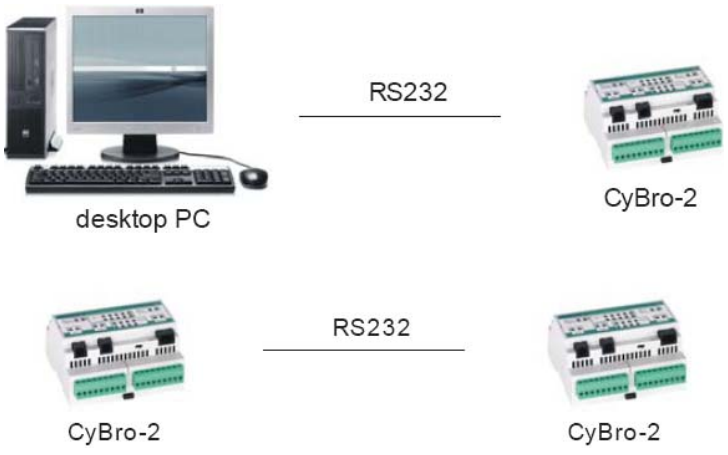
Последовательная сеть

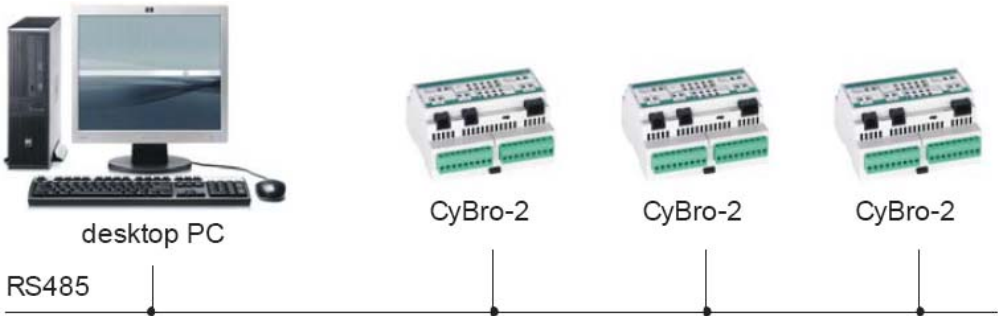
Последовательная сеть (RS-232/485) использует ведущий-ведомый (master-slave) протокол A-bus. Сетевые аппаратные средства не могут конфликтовать, так что одно устройство, ведущее в сети, ответственно за поддержку синхронизации и избегает конфликтов.

Последовательные порты CyBro соответствует стандарту RS232, который обеспечивает сквозное соединение только для двух устройств. Для присоединения трех и более устройств необходим внешний конвертор CAD-232-A2.

CAD-232-A2 является гальванически разделенным двойным конвертором RS232/RS485 с автоматическим детектированием скорости передачи и опциональным согласованием линии передачи. Это исключает источник питания из шины IEX-2, хотя это не устройство IEX-2. Два CyBro может совместно использовать один конвертор или второй вход RS232 может быть освобожден для программирования, тестирования или диагностики.

Приведенные рисунки показывают некоторые опции соединения:





Последнее соединение требует конверторов CAD-232-A2.

Последовательные гнезда

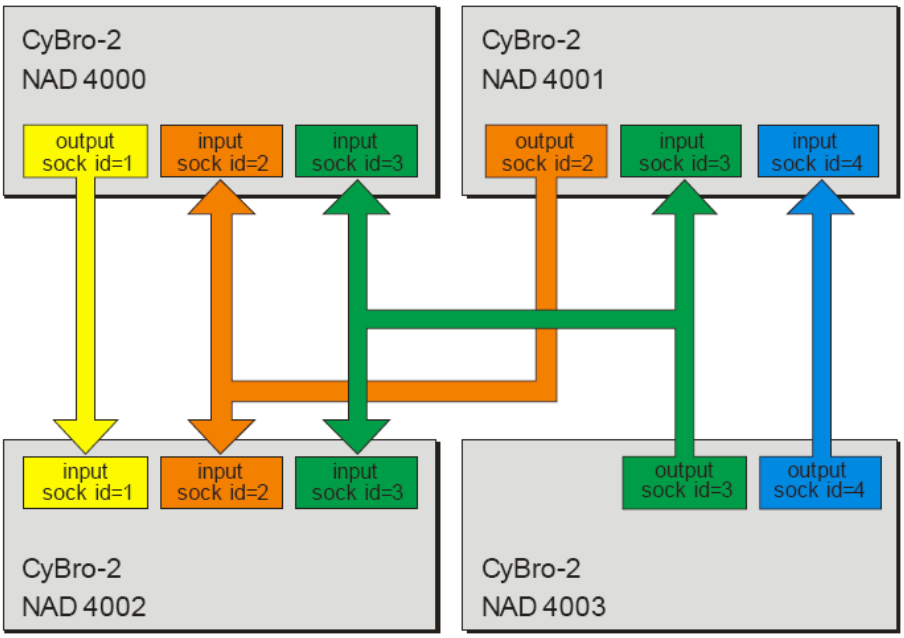
Гнездо являются устройством для коммуникации CyBro-Cybro. Оно представляет собой набор переменных, которые CyBro передает и получает через сеть.

выходное гнездо
nad 4000

входное гнездо
nad 4002

CyBro может содержать несколько входных и несколько выходных гнезд. Максимальное число 64, считая как входные, так и выходные гнезда. Каждое гнездо может содержать до 256 переменных.

Пример возможной конфигурации гнезда:



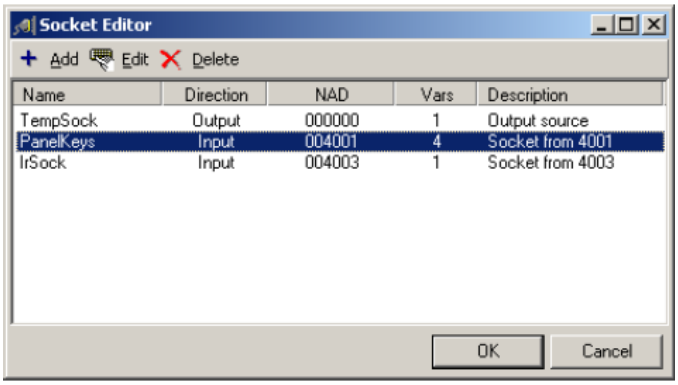
Гнездо идентифицируется id номером. как входное, так и выходное гнездо должно иметь одинаковое id. Число для свободного выбора внутри диапазона 1-255.

Согласованные гнезда должны иметь одинаковую структуру относительно типа и разрядности. Имена переменных могут различаться, хотя это не является хорошим стилем программирования.

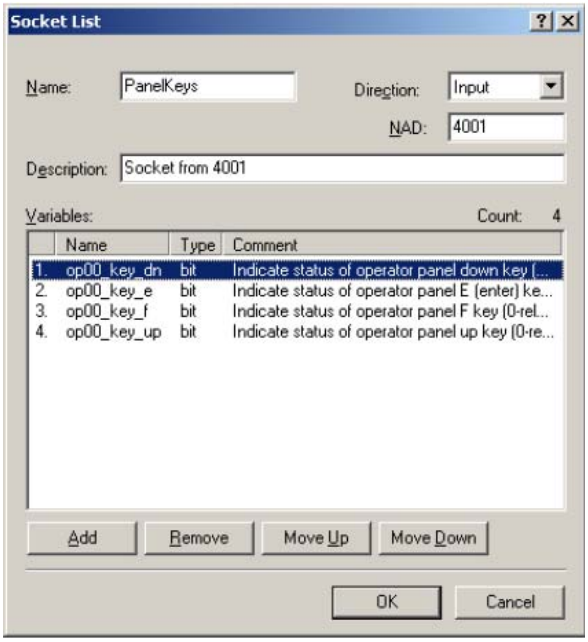
Выходное гнездо посылает широковещательный адрес и каждый CyBro в сети будет получать его. Если определяется гнездо с тем же id, оно будет использоваться для получения данных. Если такое гнездо не существует, полученные данные будут игнорироваться.

Формирователь характеристики A-bus не имело номеров id. Для сохранения совместимости с существующей сетью используйте только гнездо id 1.

Для показа списка гнезд, выберите Program/Socket Editor (программа/редактор гнезда) (F8).



Для добавления нового гнезда нажмите кнопку Add (добавить). Для редактирования выделенного гнезда нажмите кнопку Edit (редактирование).



Для ввода новой переменной нажмите кнопку Add (добавить). Чтобы изменить разрядность переменной используйте «перетаскивание» или кнопки Move Up/Move Down (движение вверх / движение вниз).

Первые два гнезда в списке могут иметь динамический номер гнезда, выбираемый из программы контроллера (PLC). Номер гнезда определяется переменными socket_0_id и socket_1_id.

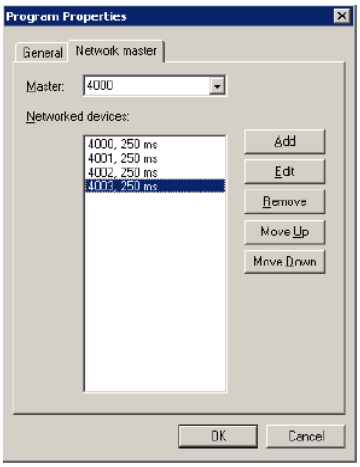
Сетевое ведущее устройство

Сетевое ведущее устройство является устройством, которое обеспечивает синхронизацию в последовательной сети. Ведущее устройство один за другим подсчитывает ведомые устройства, давая им разрешение посылать сообщения по двунаправленному каналу или команду.

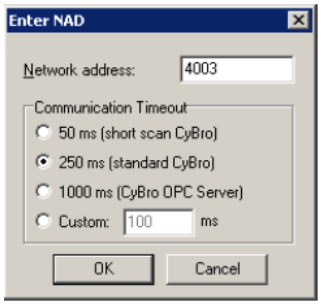
Для создания сетевого ведущего устройства откройте Program/Properties/Network master (программа/свойства/сетевое ведущее устройство) и выберите желаемый NAD (адрес).

Последовательная сеть должна иметь одно сетевое ведущее устройство, все другие устройства должны быть ведомыми.

Ведущее устройство должно иметь список устройств, которые посылают данные (имеют, по крайней мере, одно выходное гнездо). Устройства, которые только получают данные (имеют только входные гнезда), не входят в список.



Для добавления нового устройства в список нажмите кнопку Add (добавить). Появляется диалоговое окно. Введите NAD (адрес) нажмите OK.



В большинстве случаев по умолчанию время превышения назначается в 250 мсек. Только когда каждый CyBro в сети имеет очень короткую программу (менее 5 мсек), время превышения коммуникации может быть укорочено.

Слишком малое время превышения может быть причиной конфликтов в сети, выливающееся в очень большое время отклика. Если время скана CyBro больше 25 мсек, время превышения должно быть увеличено до 500 мсек или больше.

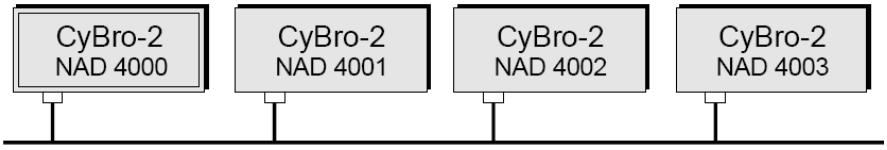
Соответствующее время превышения для CyBro OPC сервера составляет 1000 миллисекунд.

Ведущее-ведомое (master-slave) устройство не имеет отношения к направлению данных. Ведущее (master) устройство определяет разрядность и синхронизацию сообщений сети. Каждое устройство CyBro в сети, включая само ведущее устройство, может посылать и/или получать данные.

Сетевое ведущее устройство может запускаться, используя, либо переключатель работа/остановка, либо командой Start Master (запуск ведущего устройства). Стандартная команда Start запускает программу контроллера (PLC), но ведущее устройство остается неактивным.

Сетевое ведущее устройство может останавливаться, используя, либо переключатель работа/остановка, либо командой Stop Master (остановка ведущего устройства). Если команда Stop Master (остановка ведущего устройства) не работает, должен использоваться переключатель работа/остановка.

Сетевое ведущее устройство



В этом примере сетевым ведущим устройством является 4000 и ведомыми устройствами являются 4001, 4002 и 4003.

Приведенная таблица суммирует доступные режимы работы ведущих и ведомых устройств:

Устройство	Ведущее	Ведомое
CyBro Com1	+	+
CyBro Com2	-	+
CyPro	+	-
CyBro OPC сервер*	+	+
Панель управления UniOP*	+	+

* Ограниченная функциональность – выполняется только коммуникация ведущий-ведомый. Нет поддержки для коммуникации между ведомыми устройствами (обмен между гнездами). Когда необходима коммуникация между ведомым и ведомым устройством, должно использоваться ведущее устройство CyBro.

Средства отладки

Если сеть A-bus не работает, как ожидается, первым шагом должна быть запись сетевого трафика.

Чтобы сделать это загрузите AbView, монитор/вьювер A-bus, присоединяют и запускают. Оставляют запись на несколько минут и проверяют результирующий файл.

```
timestamp       ms command               from       to   size   message
------------ ----- -------------------- ------ -- ------ ----   --------------------------------
14:29:44:272     0 connected: 28.8.2007, COM1
14:29:45:389    30 send_perm              4285 ->   4208    3   AA 55 03 00 BD 10 00 00 70 10 ...
14:29:45:409    20 socket                 4208 ->   1000    8   AA 55 08 00 70 10 00 00 E8 03 ...
14:29:45:449    40 send_perm              4285 ->   4247    3   AA 55 03 00 BD 10 00 00 97 10 ...
14:29:45:469    20 socket                 4247 ->   1029    8   AA 55 08 00 97 10 00 00 05 04 ...
14:29:45:509    40 send_perm              4285 ->    100    3   AA 55 03 00 BD 10 00 00 64 00 ...
14:29:45:679   170 rd_random               100 ->   4247  173   AA 55 AD 00 64 00 00 00 97 10 ...
14:29:45:769    90 ack                    4247 ->    100  136   AA 55 88 00 97 10 00 00 64 00 ...
14:29:45:809    40 socket                 4285 ->   1000    8   AA 55 08 00 BD 10 00 00 E8 03 ...
14:29:45:849    40 send_perm              4285 ->   4208    3   AA 55 03 00 BD 10 00 00 70 10 ...
14:29:45:869    21 socket                 4208 ->   1000    8   AA 55 08 00 70 10 00 00 E8 03 ...
14:29:45:909    40 send_perm              4285 ->   4247    3   AA 55 03 00 BD 10 00 00 97 10 ...
14:29:45:919    10 socket                 4247 ->   1030    8   AA 55 08 00 97 10 00 00 06 04 ...
14:29:45:959    40 send_perm              4285 ->    100    3   AA 55 03 00 BD 10 00 00 64 00 ...
14:29:46:010    50 rd data                 100 ->   4247    7   AA 55 07 00 64 00 00 00 97 10 ...
14:29:46:110   100 ack                    4247 ->    100  154   AA 55 9A 00 97 10 00 00 64 00 ...
14:29:46:160    50 socket                 4285 ->   1000    8   AA 55 08 00 BD 10 00 00 E8 03 ...
14:29:46:200    40 send_perm              4285 ->   4208    3   AA 55 03 00 BD 10 00 00 70 10 ...
14:29:46:220    20 socket                 4208 ->   1000    8   AA 55 08 00 70 10 00 00 E8 03 ...
14:29:48:167     0 disconnected: 28.8.2007, COM1
```

Большая задержка между сообщениями может быть причиной отсутствия отклика или конфликта какого-нибудь вида.

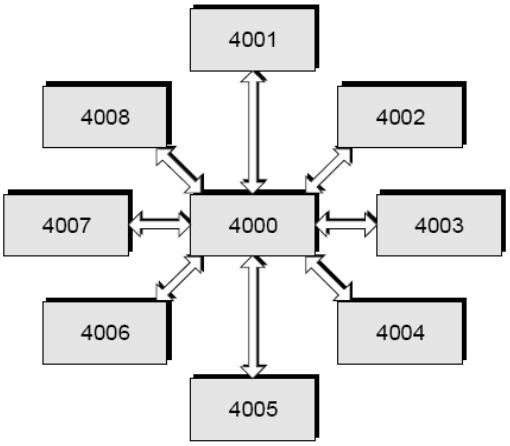
В двух случаях не может одновременно использоваться последовательный порт. Для запуска AbView войдите в CyPro или закройте последовательный порт. Если устанавливается CyBro OPC сервер, должен использоваться другой последовательный порт или компьютер.

Для того чтобы протестировать сеть Abus, может использоваться CyPro Online Monitor (оперативный монитор) как сетевое ведущее устройство. Для определения списка ведущих устройств нажмите кнопку Edit monitor mastering list (редактирование на мониторе списка ведущих устройств) и введите NADы (адрес) присоединенных устройств. Для запуска или остановки сети нажмите в Online Monitor (оперативный монитор) кнопку start/stop monitor mastering (запуск/остановка на мониторе ведущих устройств)

Тонкая настройка

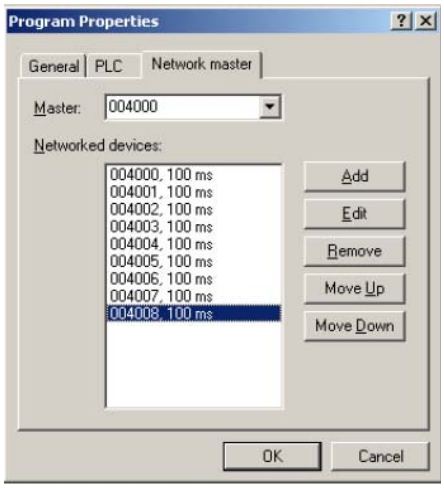
При напряженном сетевом трафике должны быть приложены усилия, чтобы добиться оптимальной характеристики сети.

Рассмотрим следующий пример:



Каждый CyBro, кроме центрального, имеет одно входное и одно выходное гнездо. Центральный CyBro имеет восемь входных и восемь выходных гнезд.

Самым коротким списком ведущих устройств будет:



Трафик между гнездами проходит в следующем порядке:

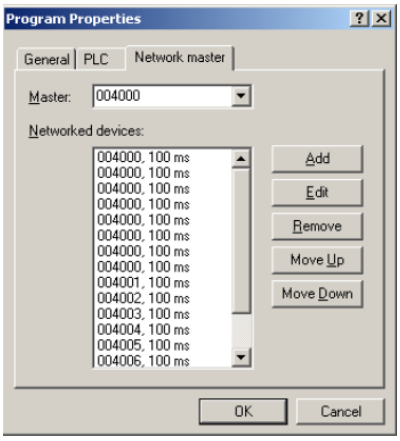
01. 4000 ⇒ 4001	10. 4000 ⇒ 4002	19. 4000 ⇒ 4003	28. 4000 ⇒ 4004
02. 4001 ⇒ 4000	11. 4001 ⇒ 4000	20. 4001 ⇒ 4000	29. 4001 ⇒ 4000
03. 4002 ⇒ 4000	12. 4002 ⇒ 4000	21. 4002 ⇒ 4000	30. 4002 ⇒ 4000
04. 4003 ⇒ 4000	13. 4003 ⇒ 4000	22. 4003 ⇒ 4000	31. 4003 ⇒ 4000
05. 4004 ⇒ 4000	14. 4004 ⇒ 4000	23. 4004 ⇒ 4000	32. 4004 ⇒ 4000
06. 4005 ⇒ 4000	15. 4005 ⇒ 4000	24. 4005 ⇒ 4000	…
07. 4006 ⇒ 4000	16. 4006 ⇒ 4000	25. 4006 ⇒ 4000	
08. 4007 ⇒ 4000	17. 4007 ⇒ 4000	26. 4007 ⇒ 4000	
09. 4008 ⇒ 4000	18. 4008 ⇒ 4000	27. 4008 ⇒ 4000	

Полный коммуникационный цикл производит 9x8=72 коммуникационных сообщений. Длительность одного коммуникационного цикла зависит от многого, но для средней нагрузки CyBro с малым гнездом, 50 мсек является довольно хорошей оценкой. В этом случае полное время обновления сети будет 72x50 мсек = 3,6 сек, что довольно значительно.

Для того чтобы добиться лучшего времени отклика, список сетевых ведущих устройств должен содержать несколько адресов для CyBro, которые имеют большее количество выходных гнезд. Число адресов должно быть равно числу выходных гнезд.

В нашем примере адрес 4000 должен повторяться восемь раз, один раз для каждого выходного гнезда. Другие адреса записываются в список, поскольку они имеют только одно выходное гнездо.

Новый список ведущих устройств показан ниже:

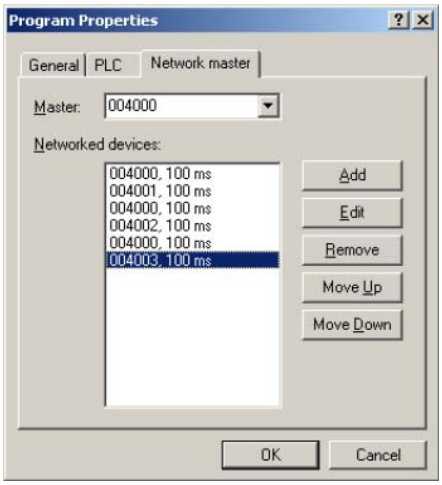


Трафик для гнезда теперь намного лучше оптимизируется:

01. 4000 ⇒ 4001	05. 4000 ⇒ 4005	09. 4000 ⇒ 4000	13. 4005 ⇒ 4000
02. 4000 ⇒ 4002	06. 4000 ⇒ 4006	10. 4002 ⇒ 4000	14. 4006 ⇒ 4000
03. 4000 ⇒ 4003	07. 4006 ⇒ 4007	11. 4003 ⇒ 4000	15. 4007 ⇒ 4000
04. 4000 ⇒ 4004	08. 4007 ⇒ 4008	12. 4004 ⇒ 4000	16. 4008 ⇒ 4000

Все гнезда используются в 8+8=16 коммуникационных циклах. Сетевое время отклика падает вниз до 16x50 мсек = 800 мсек, быстрее в 4,5 раза.

Для увеличения приоритета также может использоваться несколько адресов. Следующий пример показывает сеть с увеличенным приоритетом для выходного гнезда 4000:



На пропускание сети также может оказывать воздействие сетевого ведущего устройства. Для лучшего результата сетевое ведущее устройство должно иметь короткую программу контроллера (PLC), поскольку большое время скана может быть причиной задержки между коммуникационными сообщениями.

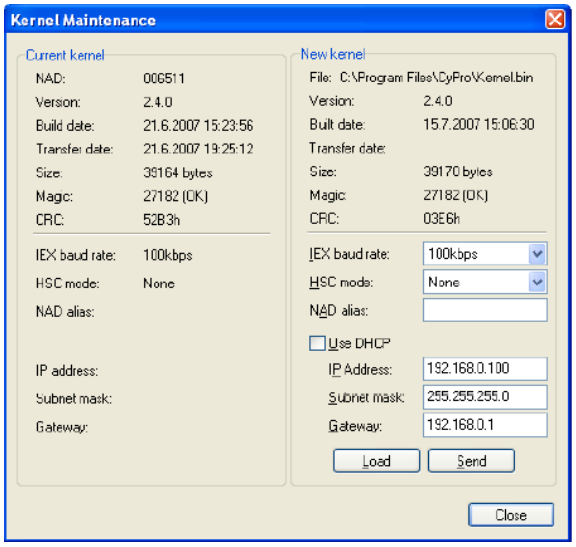
Также можно добиться некоторого улучшения, когда сетевым ведущим устройством является CyBro с большим количеством выходных гнезд, поскольку выходное гнездо ведущего устройства немного быстрее, чем гнездо ведомого устройства.

Ethernet

Сеть TCP/IP использует A-bus протокол с многими ведущими устройствами. Сетевое оборудование управляет конфликтами, так что каждое устройство может посылать коммуникационное сообщение в любой момент.

Каждое TCP/IP устройство имеет 4-байтовый сетевой адрес называемый IP адресом. CyBro может иметь динамический IP адрес, используя внешний DHCP сервер или статический IP адрес, используя предварительно конфигурированный фиксированный адрес.

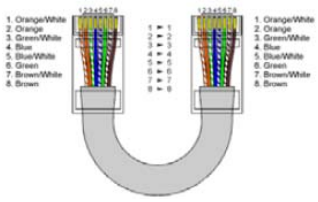
Для определения статического адреса откройте Kernel Maintenance (обслуживание ядра), отключите кнопку-флажок Use DHCP и введите IP адрес, Subnet mask (маска подсети) и Gateway (шлюз). Если любой адрес неправильный, кнопка Send (посылка) блокируется.



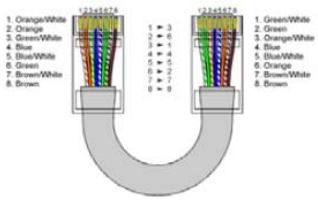
После включения питания статический IP адрес немедленно разблокируется. Динамический IP адрес необходим некоторое время, обычно 2-5 секунд в той же сети при последней загрузке или 30-35 секунд в новой сети с другой маской подсети.

Если доступен DHCP сервер, CyBro не будет иметь IP адреса (0.0.0.0). Хотя это неразрешенный коммуникационный адрес, он возможен для присоединения с использованием коммуникационной настройки Direct connection (прямое соединение).

CyBro имеет стандартный коннектор RJ-45 UTP. Скорость передачи 10/100 Мб, автодетектируемая.



8-жильный соед. кабель



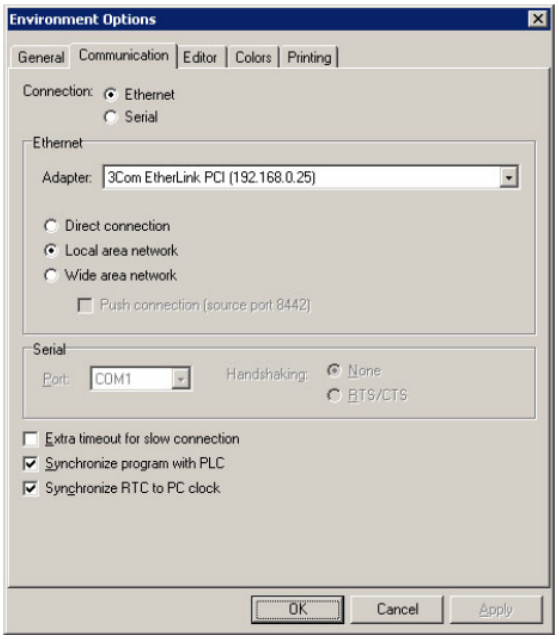
8-жильный кроссоверный кабель

Соединительный кабель может быть стандартным соединительным кабелем или кроссоверным кабелем, также автоматически детектируемым.

Каждый CyBro имеет уникальный 6-байтовый аппаратный адрес (MAC адрес) в форме 00-CB-00-xx-xx-xx, где последние три байта являются серийным номером (NAD (адрес)) CyBro. Например, CyBro 6512 (00196AF hex) имеет MAC адрес 00=CB-00-00-19-6F.

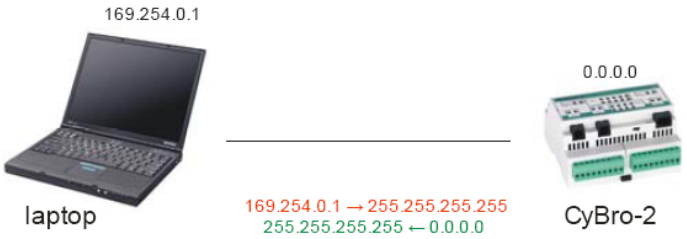
Опции соединения

Поскольку TCP/IP универсально, CyBro и ПК может использовать в основном три различных соединения: Direct connection (прямое соединение), Local area network (LAN) и Wide area network (интернет).



Если компьютер имеет два или больше сетевых адаптеров, правильный адаптер должен выбираться вручную.

1. Прямое соединение



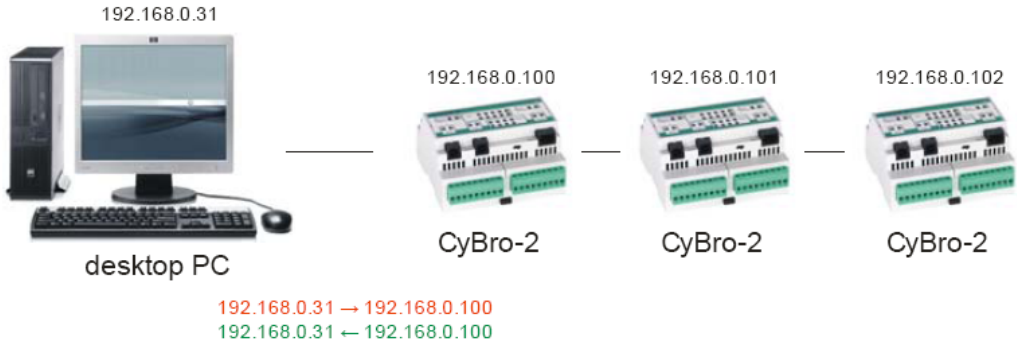
Это соединение используется, когда соединяются только два устройства ПК и CyBro.

Коммуникация использует ограниченный широковещательный адрес (255.255.255.255:65535).

При отсутствии DHCP сервера ПК использует автоконфигурируемый адрес (169.254.0.1-169.254.255.255). CyBro будет иметь либо адрес (0.0.0.0), либо первоначально назначенный статический адрес. Оба способа обеспечивают для соединения широковещательный адрес.

Не используйте эту настройку для локальной сети. Ограниченный широковещательный адрес не отправляется маршрутизатором, так что только часть сети доступна.

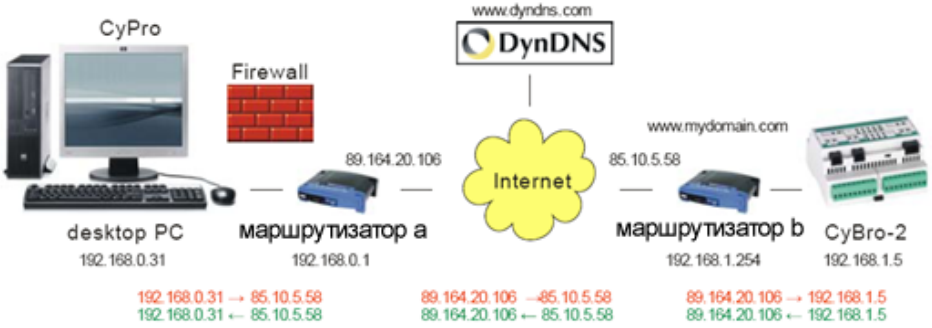
2. Локальная сеть



Это обычная настройка для малой домашней или офисной сети. Все компьютеры, серверы и контроллеры совместно используются под одним и тем же адресом подсети. Адресация может быть динамичной (используя DHCP сервер) или статичной.

CyPro будет автоматически детектировать каждый IP адрес, используя прямой широковещательный адрес (192.168.0.255).

3. Глобальная сеть



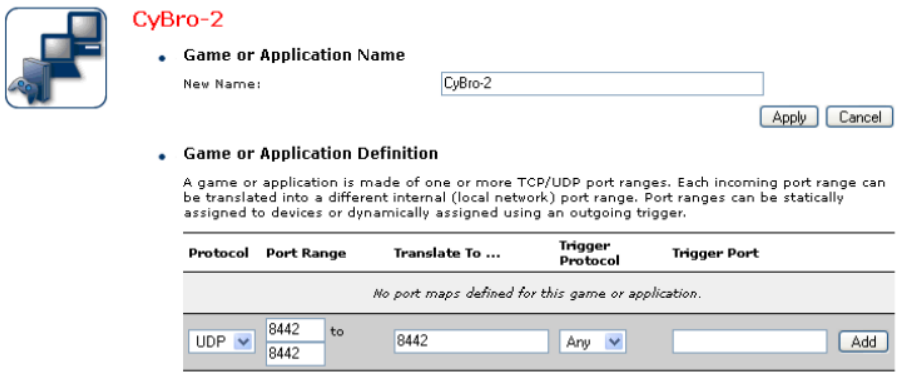
Это соединение обеспечивает программирование/мониторинг в зоне Интернета. Для настройки всех устройств должны выполняться ряд шагов:

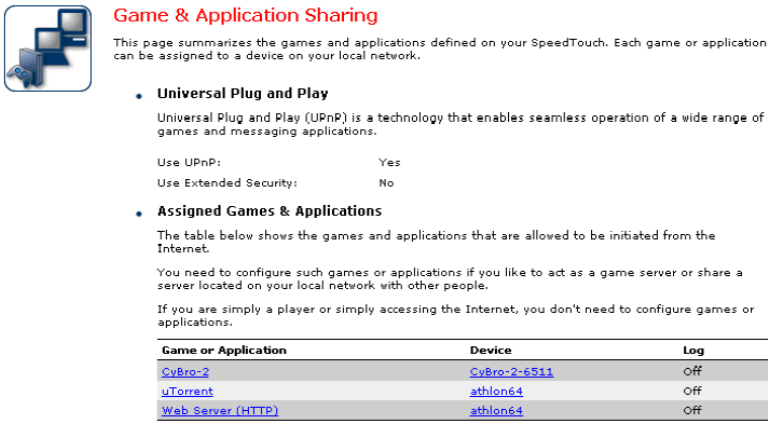
1. CyPro — Опции, настройка соединения Ethernet в глобальной сети. Введите WAN адрес (85.10.5.58) маршрутизатора b в поле Address (адрес).

2. Брандмауэр — Разблокирует весь исходящий трафик и открывает входящее соединение UDP от порта 8442. Для проверки можно временно заблокировать брандмауэр.

3. Маршрутизатор a — Нет необходимости в специальных настройках. Убедитесь, что исходящий трафик не блокирован.

4. Маршрутизатор b — Назначьте входящему UDP порту 8442 IP адрес CyPro:





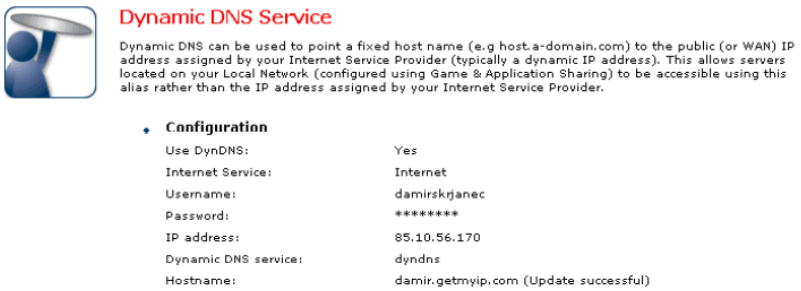
Если в локальной сети более одного CyBro, назначьте порту 8442 локальной широковещательный адрес 192.168.0.255. Если маршрутизатор не допускает этого, используйте отдельный входной порт для каждого CyBro (например, 10000-10002). В этом случае используйте тот же номер порта для CyPro Ethernet адрес (85.10.5.58:10000).

5. CyBro-2 Никаких специальных настроек не требуется. Может быть использован DHCP или статический IP.

В типичной домашней сети, соединение в которой сделано через ADSL, фиксированный IP адрес может быть недоступен. Как вариант может быть использован динамический DNS сервис.

6. DNS сервис Зарегистрируйтесь в динамическом DNS сервисе, таком как www.dyndns.com. Выберите домен. Некоторые домены (например, xxx.getmyip.com) доступны бесплатно.

7. DNS клиент Конфигурируйте динамический DNS клиент на маршрутизатор b:



Если маршрутизатор не имеет динамической DNS поддержки, клиент может устанавливаться на местный ПК.

Для пошагового тестирования соединения используйте отправитель пакетов (PING) или программу контроля прохождения сигнала.

Когда соединение сделано, дистанционная работа проходит так же как в локальной сети. Автодетектирование аппаратных средств, посылка программы, оперативный монитор, обслуживание ядра – все функции доступны.

Гнезда Ethernet

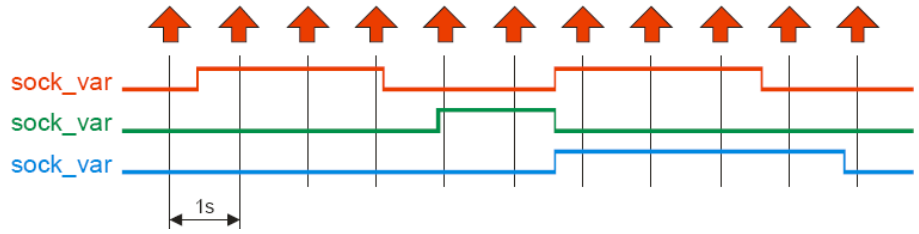
Гнездо обмена Ethernet подобно серийному порту, но с важной разницей: нет сетевого ведущего устройства. Каждый CyBro заботится о выходных гнездах. В отличие от последовательной коммуникации, где гнезда непрерывно производят передачу, гнезда Ethernet производят передачу только, когда это необходимо. В результате получается короткая задержка и лучшее использование пропускной способности.

Как и в последовательной коммуникации входное и выходное гнезда согласуются по id номеру. Каждое выходное гнездо также является входным гнездом, там нет особых выходных гнезд.

Гнезда работают на передачу только, когда контроллер (PLC) работает.

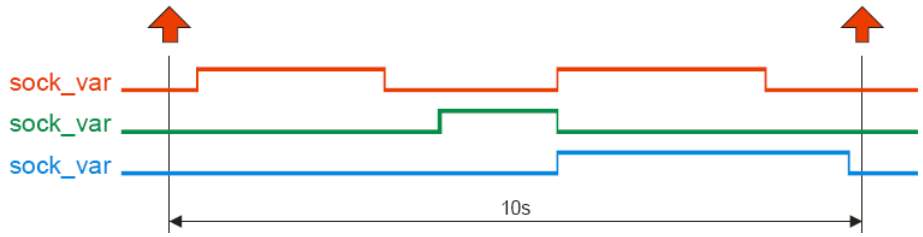
Для каждого выходного гнезда может быть выбран один из четырех режимов выхода:

1. Периодичность 1 сек



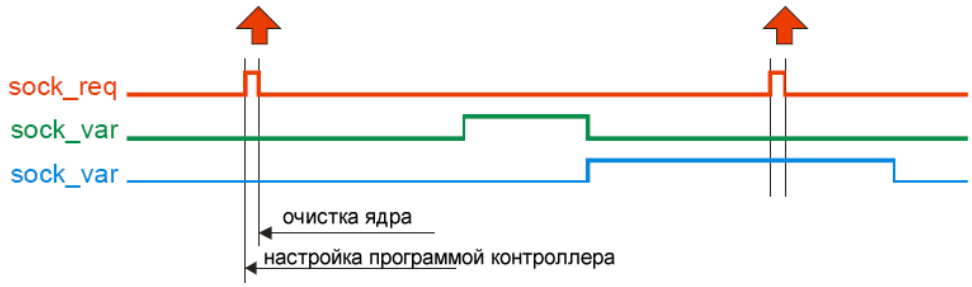
Гнездо периодически работает на передачу, один раз в секунду.

2. Периодичность 10 сек



Гнездо периодически работает на передачу, один раз в 10 секунд.

3. По запросы



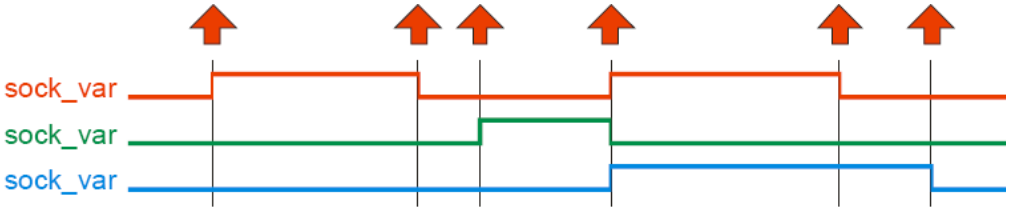
Запросное гнездо периодически работает на передачу.

Передача начинается, когда программа контроллера настраивает требуемый бит. Ядро реагирует очисткой запроса и отсылкой в гнездо.

Бит запроса является первой битовой переменной в гнезде. По соглашению бит запроса всегда посылается как 1. Гнездо получения не будет снова производить повторную передачу. Запрос автоматически удаляется после скана.

Для инициации передачи внешним устройством (например, другим CyBro) используют второе гнездо. Переменная запроса не может быть общей, поскольку она автоматически удаляется после приема.

4. На замену



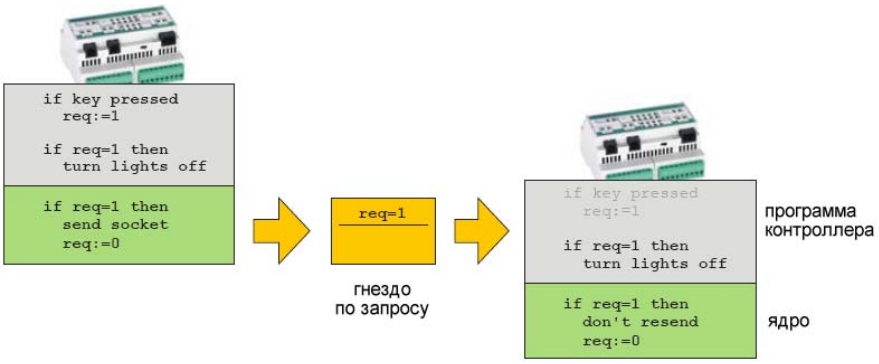
Гнездо передает каждый раз при изменении переменной гнезда. Гнездо получения не будет снова производить повторную передачу.

Примеры гнезд

Событийно-управляемое действие

Запросное выходное гнездо может использоваться для посылки единичного события в сеть. Каждый контроллер может инициировать событие, каждый контроллер будет получать событие и запрос будет автоматически переустанавливаться. Число контроллеров не ограничивается.

Пример показывает событие выключения всех источников света.

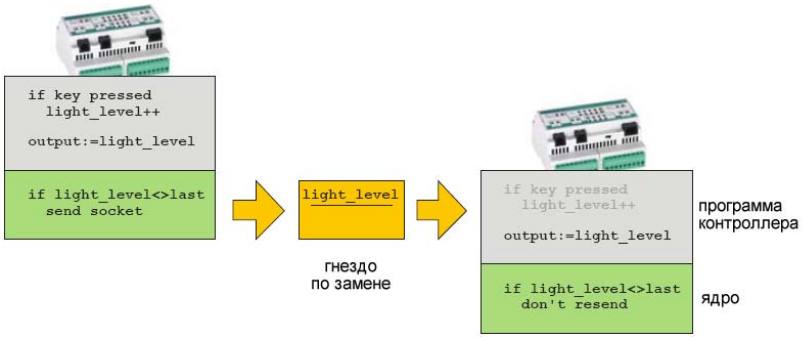


Каждый CyBro может иметь подобную программу, хотя назначение местного вх/вых не может быть тем же самым. Структура программы является простым, поскольку требуется управление полностью автоматическое – один раз активированная сеть заботится, во-первых, о распространении запроса и затем переустанавливает их.

Синхронизированное значение

Выходное гнездо по замене может использоваться для синхронизации общего значения для нескольких контроллеров. Каждый контроллер может модифицировать значение и каждый контроллер будет получать последнее модифицированное значение. Число контроллеров не ограничивается.

Пример показывает общую настройку освещения (0-100%) в большом зале.



Каждый CyBro имеет одинаковую программу, хотя назначение местного вх/вых может быть различным. Поскольку синхронизация автоматическая, структура программы очень проста.

Дополнительные особенности

MODBUS

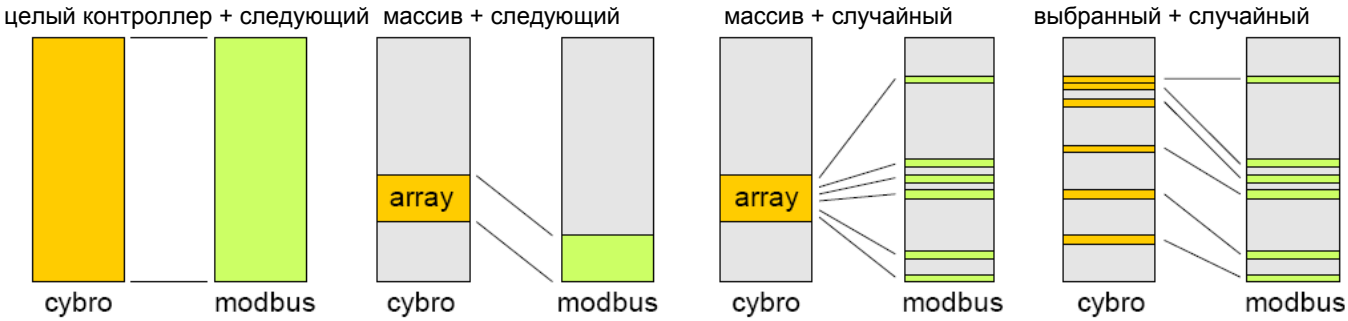
Modbus является протоколом последовательной коммуникации предложенным Модиконом в 1979 году для использования его с программируемыми логическими контроллерами (PLC). Он стал на самом деле стандартным протоколом коммуникации в промышленности и он является теперь наиболее общими доступными средством для присоединения промышленных электронных устройств. Modbus разрешает коммуникацию между многими устройствам присоединенными к одной сети.

CyBro поддерживает:

- Ведомое устройство (RS232 или RS485) Modbus MTU
- Ведомое устройство (Ethernet) Modbus TCP

Ведущее устройство Modbus в настоящее время не поддерживается.

Модель данных Modbus описывает как трансформаторы и реестры Modbus транслируются в память CyBro.

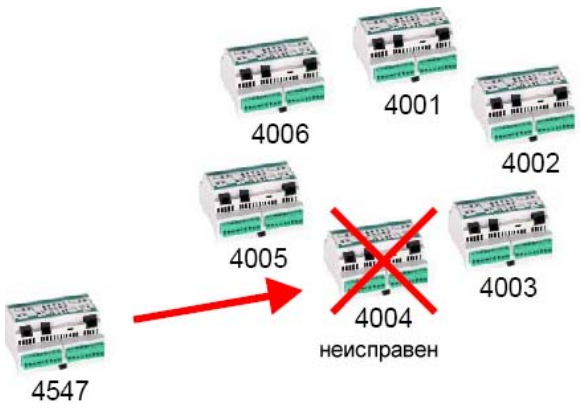


Выбор соответствующей модели ведет к короткой и эффективной программе контроллера.

Альтернативный NAD (адрес)

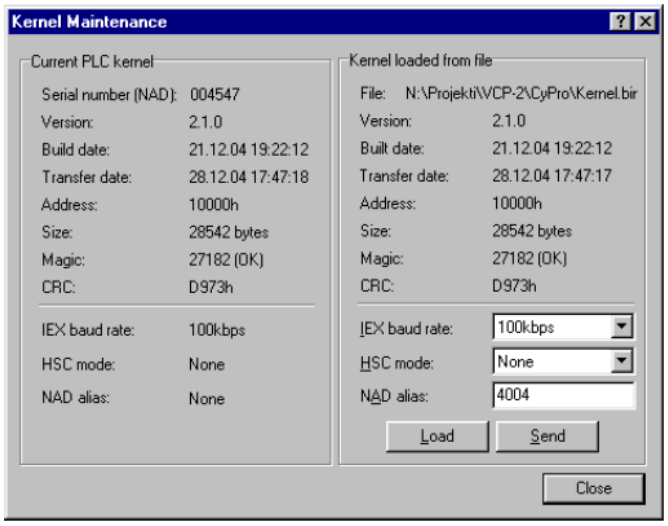
Каждый блок CyBro имеет уникальный серийный номер, используемый также как коммуникационный адрес (NAD (адрес)). Серийный номер непрерывный и не может быть изменен.

Альтернативным NAD (адрес) является вторым коммуникационным адресом конфигурируемым пользователем. Установленный один раз альтернативный адрес функционирует как оригинальный NAD (адрес). CyBro с альтернативным адресом может работать в сети, как с серийным номером, так и с альтернативным адресом.



альтернативный адрес NAD (адрес) = 4004

Для ввода альтернативного адреса NAD (адрес) откройте Kernel Maintenance (обслуживание ядра), введите желаемый альтернативный адрес и пошлите его в ядро.

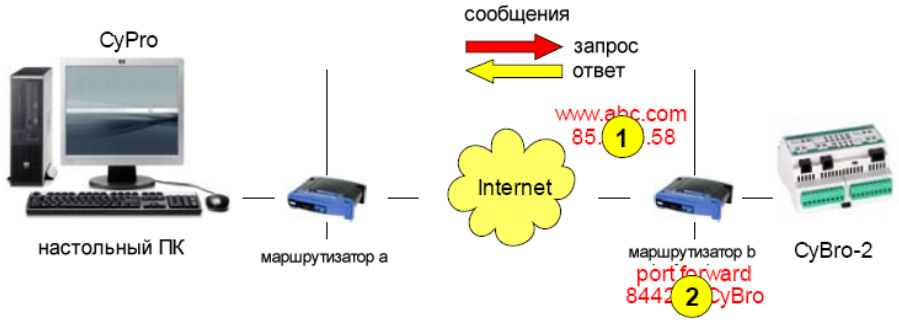


Для определения действительных адресов откройте диалоговое окно Get PLC Info (получение информации о контроллере). Загруженная таблица показывает оригинальный NAD (адрес) (серийный номер) и таблица ядра (Kernel) показывает альтернативный адрес.

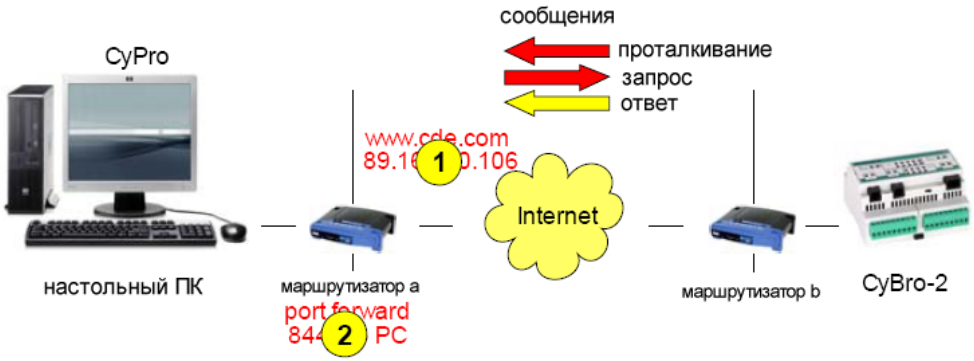
Соединение проталкивания

Соединение проталкивания является альтернативным способом встройки CyBro в Интернет.

В стандартном соединении (описанной в Connection options (опции соединения)) необходим WAN адрес маршрутизатора b и конфигурированный порт продвижения:



В соединении проталкивания CyBro необходим WAN адрес и роутер a должен конфигурироваться в направлении сообщений к ПК:

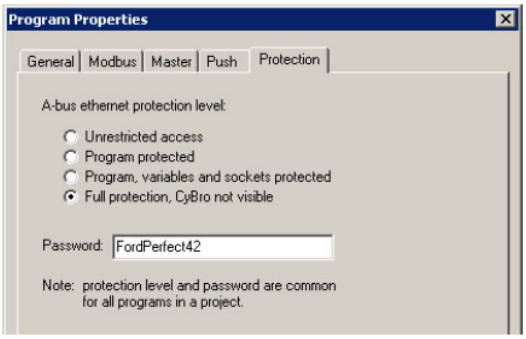


В некоторых случаях это возможное соединение может быть более удобно, чем стандартное соединение.

Для проверки соединения проталкивания загрузите CyBroPushServer откройте пример PushDemo и прочитайте инструкции.

Защита паролем

Начиная с версии 2.5.0, CyBro имеет функцию ограничения доступа паролем. В зависимости от выбранного уровня защиты, она может распространяться на программу контроллера (PLC), переменные и гнезда. Например, если уровень защиты является Program protected (программа защищена), любой человек может свободно считать и записать параметры, но будет требоваться пароль для посылки новой программы.



Защита паролем эффективна только для Ethernet. Последовательный доступ не ограничен даже, если CyBro конфигурируется для полной защиты.

Пароль может содержать любую комбинацию букв и чисел значительной длины. Не используйте пробел, специальные значки или национальные символы.

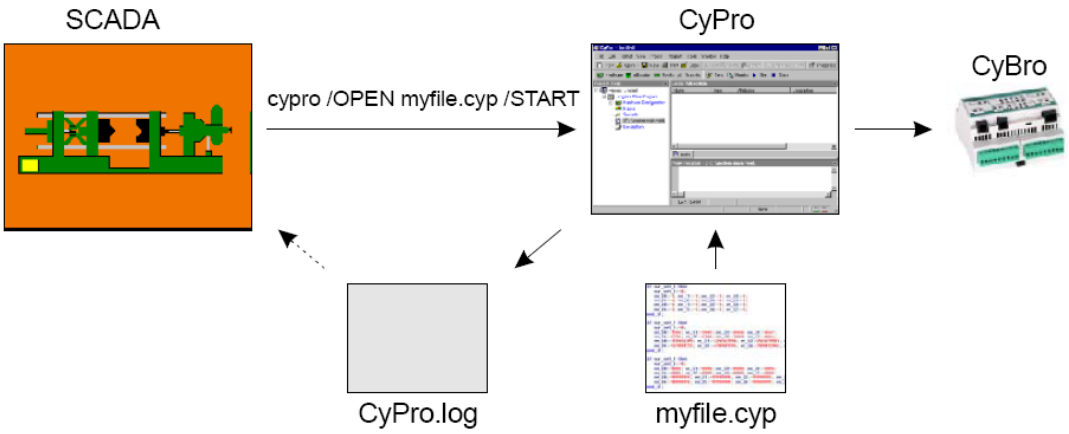
Пароль общий для всех программ, нет возможности определить отдельный пароль для каждого CyBro. Пароль, хранимый в файле проекта, не защищен, так что держите в безопасности файл проекта.

Для посылки нового пароля используйте команду Erase protected program (стирание защищенной программы).

Опции строки команд

Опции командной строки могут указываться при запуске CyPro. Они используются для автоматического выполнения некоторых заданий, таких как посылка программы в присоединенный контроллер (PLC).

Используя опции строки команд, CyPro может использоваться как внешний компилятор для пользовательского интерфейса.



Для использования опций командной строки есть два обычных стилевых решения:

cypro.exe filename.cyp
cypro.exe /OPTION1 /OPTION2 /OPTION3…

Первое стилевое решение используется работающей системой при двойном клике пользователя на файле cyp.

Опции командной строки:

/NEW [filename.cyp]	Создание нового проекта. Имя файла опциональное. Если имя пропускается, вместо этого используется “untitled” (без названия).
/OPEN filemane.cyp	Открытие существующего проекта с указанным именем файла.
/SAVE	Сохранение проекта.
/SAVEAS filename.cyp	Сохранение проекта под указанным именем.
/EXIT	Выход из CyPro.
/NAD number	Выбор программы. Если указанный NAD (адрес) существует, будет выбираться эта программа, в другом случае NAD (адрес) присоединяется к текущей программе.
/AUTODETECT	Автодетектирование аппаратных средств.
/START	Компиляция, посылка (только если отличается) и запуск.
/STARTALL	Запуск всех программ проекта.
/STOP	Остановка текущей программы.
/SEND	Посылка текущей программы.
/HIDDEN	Скрытая работа, не показывается никаких окон или диалогов.

Имя файла может быть просто именем или полным путем. Если имя файла содержит пробел, должны использоваться квоты (“my file.cyp”). Если для продолжения выполнения работы программы требуется пользовательский вход, автоматически будет использоваться опция по умолчанию. Например, когда автодетектировании запрашивается сетевой адрес, адрес по умолчанию (нуль) будет использоваться автоматически.

При запуске с опций строки команд CyPro создает лог файл “CyPro.log”, который содержит все заданные команды и их результаты (доступ или отказ). Лог файл сохраняется в директории CyPro (c:\Program Files\CyPro\CyPro.log).

Если используется режим /HIDDEN, CyPro будет автоматически заканчивать работу после последней выполненной команды.

При использовании опций командной строки советуем включить кнопку-флажок Allow multiple instances (допускается несколько копий) в Environment Options (опции окружающей среды). Если допускается только одна копия и CyPro уже работает, запросы командной строки будут предшествовать активному копированию.

Примеры:

```
cypro.exe  myfile . cyp
```

Запускает CyPro и открывает проект myfile.cyp.

```
cypro.exe  “c:\My   Document \myfile . cyp”
```

Запускает CyPro и открывает проект myfile.cyp в указанной директории. Так как путь может содержать пробел, требуется квота.

```
cypro.exe  /HIDDEN / OPEN “myfile . cyp” /START /EXIT
```

Запускает CyPro, открывает существующий проект (myfile.cyp), запускает контроллер (PLC) (компилирует, посылает и работает) и заканчивает работу. Работа происходит скрытно, никаких окон или диалогов не будет появляться. Возможные ошибки сохраняются в CyPro.log.

```
cypro.exe  /HIDDEN / NEW /AUTODETECT /SAVEAS “myfile . cyp” /EXIT
```

Запускает CyPro, открывает новый проект, запускает автодетектирование, сохраняет как myfile.cyp и завершает работу. Работа происходит скрытно, никаких окон или диалогов не будет появляться. Возможные ошибки сохраняются в CyPro.log.

```
cypro.exe  /HIDDEN / OPEN “myfile . cyp” /START /EXIT
```

Запускает CyPro, открывает новый проект, добавляет новый NAD (адрес), запускает автодетектирование для определение присоединенных модулей IEX-2, сохраняет как myfile.cyp и завершает работу. Работа видна, никаких окон или диалогов не будет появляться. Возможные ошибки сохраняются в CyPro.log.

```
cypro.exe  /HIDDEN / OPEN “myfile . cyp” /AUTODETECT / START /EXIT
```

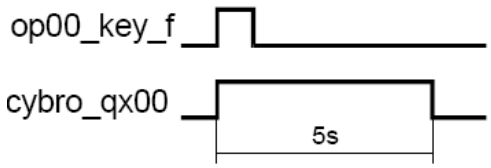
Запускает CyPro, открывает существующий проект (myfile.cyp), запускает автодетектирование (предполагая, что проект не имеет определенную настройку аппаратных средств и сетевой адрес, как в примерах CyPro), запускает контроллер (PLC) (компилирует, посылает и работает) и заканчивает работу. Оригинальный файл будет оставаться неизмененным. Работа видна, никаких окон или диалогов не будет появляться. Возможные ошибки сохраняются в CyPro.log.

Учебное пособие по CyBro

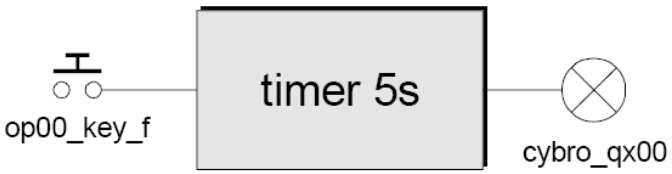
Ваша первая программа для CyBro

Первый шаг: определение проблемы

В первом примере мы возьмем очень простое задание: таймер активируемый нажатием кнопки. При нажатии кнопки таймер будет включать выход не предопределенный период времени около 5 секунд.



Кнопкой для активации таймера может быть любой двоичный вход, проще использовать одну из кнопок дисплея. Выход представляет собой первый двоичный выход cybro_qx00.

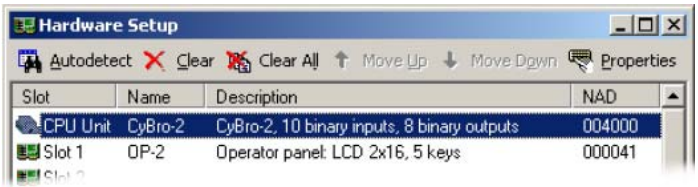


Второй шаг: выбор аппаратных средств

Соедините CyBro и ПК последовательным кабелем.

Запустите CyPro и для запуска нового проекта выберите File/New Project (файл/новый проект). Откройте Hardware Setup (настройка аппаратных сред) и запустите Autodetect (автодетектирование). Появится малое диалоговое окно, запрашивающее ввод адреса CyBro.

Если присоединяется один CyBro (не сеть), нажмите OK. Процедура автодетектирования будет идентифицировать все присоединенные модули IEX-2. В нашем примере будет определяться CyBro и панель управления OP-2.

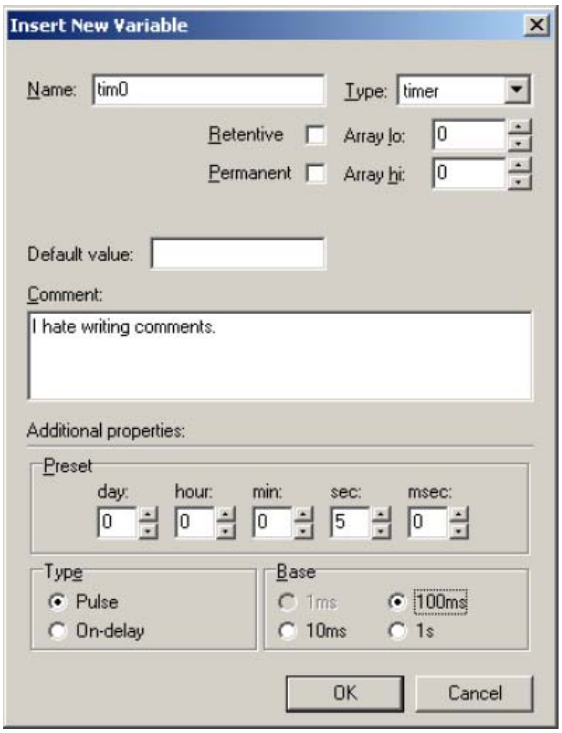


Для доступа к автодетектированному оборудованию кликните на кнопке OK.

Третий шаг: распределение переменных

Для нашего простого примера необходима одна переменная типового таймера.

Запустите Allocation Editor (редактор распределения) и нажмите Insert (вставить) в Insert New Variable (вставить новую переменную). Появится диалоговое окно:



Введите имя, выберите тип, настройте предустановленное значение, выберите тип импульса, выберите в базе 100 мсек и нажмите OK.

Четвертый шаг: запись кода

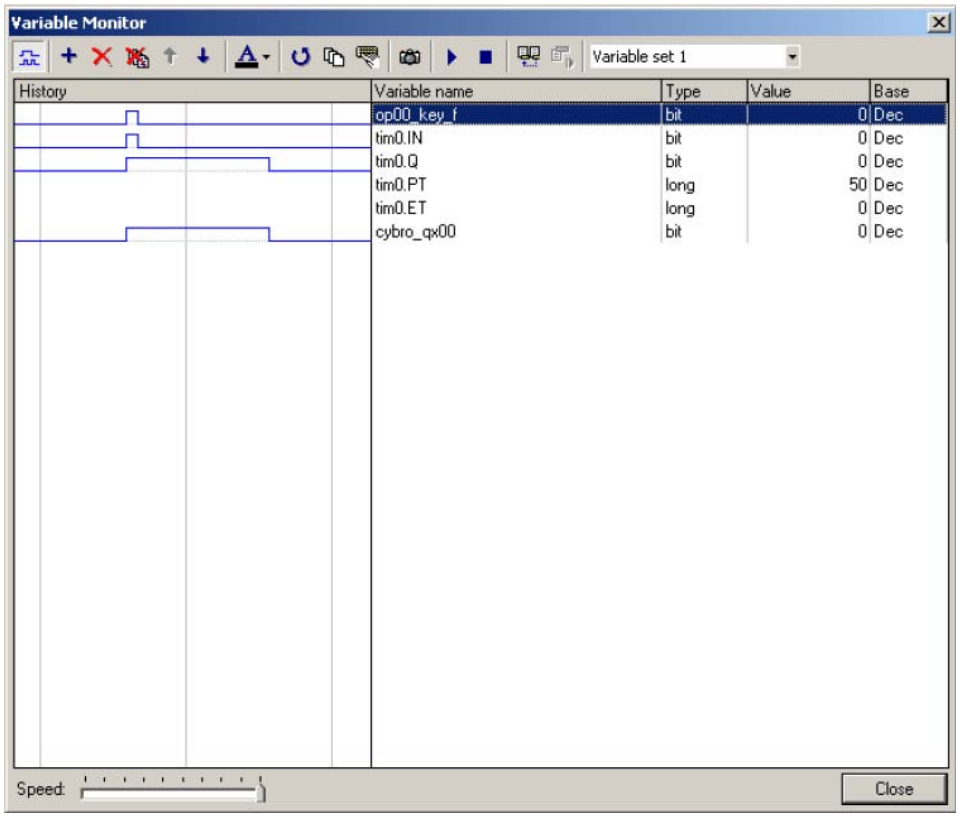
Код контроллера (PLC) должен присоединить вход таймера к ключу и выход таймера к выходному реле. Это можно сделать:

```
tim0 . in : =op00      key    f;
cybro_qx00 : =tim . q;
```

Пятый шаг: доведение программы до жизненных условий

Чтобы компилировать и передать программу в CyBro, нажмите кнопку Start (запуск). Статусная линия индикатора будет показывать, что программа работает.

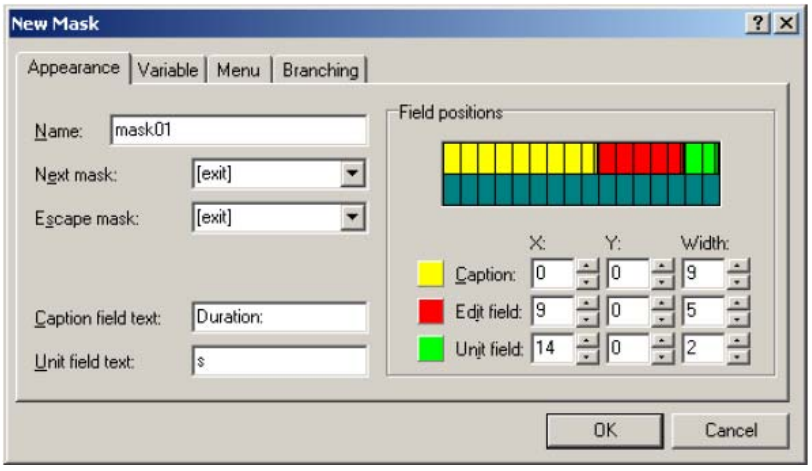
Чтобы проверить всю работу, запустите Variable Monitor (монитор переменной) и Add (добавить) назначенные параметры.



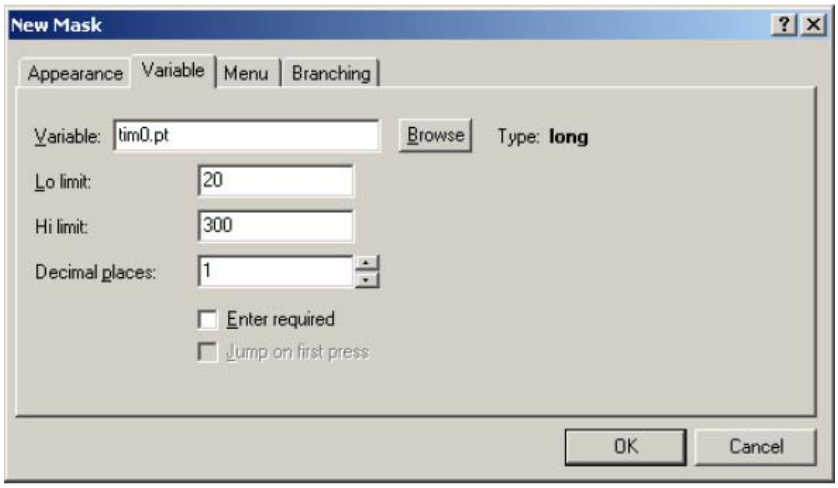
Шестой шаг: новые границы

Другая проблема должна быть решена с настройкой таймера.

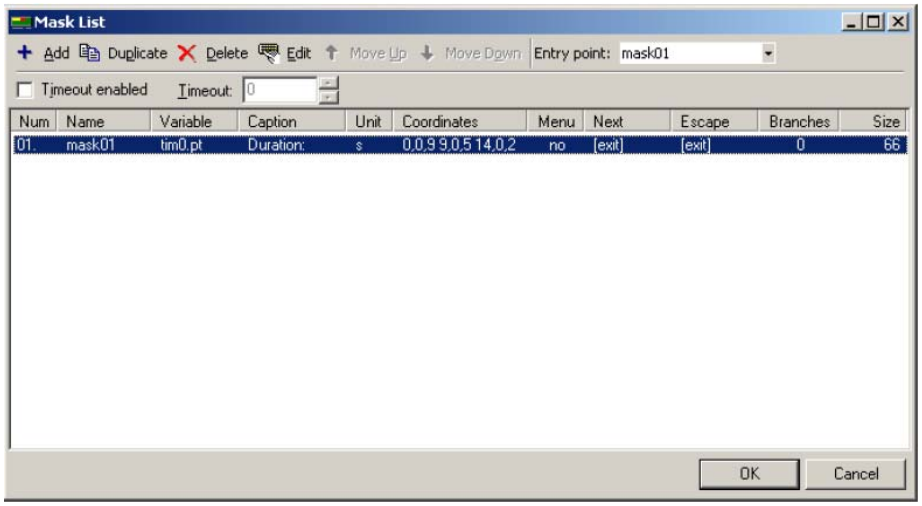
Способом решить ее является система масок. Запустите Mask Editor (редактор маски) и нажмите Add (добавить) для создания новой маски. Введите заглавие и тексты полей блоков. Другие поля могут оставаться на значениях по умолчанию.



Переключитесь на закладку Variable (переменная), введите переменную tim0.pt и настройте границы. Заметим, что разрешение таймера 100 мсек, так что для правильной индикации секунд требуется одна позиция десятичной точки.

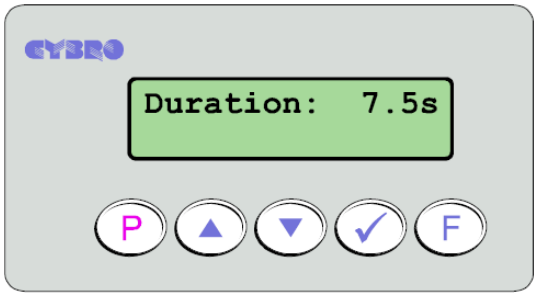


Нажмите OK. Настройте ввод точки в mask01 и опять нажмите OK.



Нажмите кнопку Start (запуск). Программа автоматически компилируется, передается и запускается.

Для настройки значения таймера нажмите кнопку P, сделайте настройку кнопками up (вверх) и dn (вниз) и выйдите повторным нажатием на P.



Для активации таймера нажмите кнопку F.

Приложение

Сводка типов данных

Единичные

тип	разрядность	диапазон
двоичный	1	0..1
слово	16	-
целое число	16	-32768..32767
длинное целое	32	-2147483648..2147483647
реальное	32	$-3,4x10^{38}..3,4x10^{38}$

Вход / Выход

тип	разрядность	равный тип	описание
двоичный вход	1	двоичный	Двоичный вход
двоичный выход	1	двоичный	Двоичный выход
вход слово	16	целое число	Аналоговый вход
выход слово	16	целое число	Аналоговый выход

Таймер

поле	тип	направление	описание
in	двоичный	двоичный	вход
pt	длинное целое	двоичный	предустановленное время
et	длинное целое	целое число	время работы
q	двоичный	целое число	выход

Счетчик

поле	тип	направление	описание
cu	двоичный	вход	вход счета вверх
cd	двоичный	вход	вход счета вниз
ld	двоичный	вход	вход нагрузки
r	двоичный	вход	вход переустановки
pv	целое число	вход	предустановленное значение
cv	целое число	вход	значение счетчика
qu	двоичный	выход	выход верхнего предела
qd	двоичный	выход	выход нижнего предела

Внутренние переменные

имя	тип	направление	описание
first_scan	бит	только чтение	активен только во время первого скана
scan_overrun	бит	только чтение	происходит превышение времени скана
clock_10ms	бит	только чтение	время фиксировано на 10 мсек
clock_100ms	бит	только чтение	время фиксировано на 100 мсек
clock_1s	бит	только чтение	время фиксировано на 1 сек
clock_10s	бит	только чтение	время фиксировано на 10 сек
clock_1min	бит	только чтение	время фиксировано на 1 мин
retentive_fail	бит	чтение/запись	показывает, что сохраняющаяся память неисправна
cybro_outputs_off	бит	чтение/запись	если активна, все двоичный входы CyBro испорчены
disconnect_inputs	бит	чтение/запись	если активна, входами можно манипулировать вручную
no_input_filter	бит	чтение/запись	если активна, местный входной фильтр 5 мсек выкл.
rtc_write_req	бит	чтение/запись	запрос записи в RTC
ee_read_req	бит	чтение/запись	настройка на чтение всех EE переменных из EEPROM
ee_write_req	бит	чтение/запись	настройка записи всех EE переменных в EEPROM
ee_write_magic	бит	чтение/запись	настройка на 31415 на разблокировку записи в EEPROM
scan_time	цел.	только чтение	время выполнения последнего скана (мсек)
scan_time_max	цел.	только чтение	ограничение макс. времени выполнения скана (мсек)
scan_frequency	цел.	только чтение	число сканов за секунду
rtc_sec	цел.	чтение/запись	RTC секунда
rtc_min	цел.	чтение/запись	RTC минута
rtc_hour	цел.	чтение/запись	RTC час
rtc_weekday	цел.	чтение/запись	RTC день недели
rtc_date	цел.	чтение/запись	RTC дата
rtc_month	цел.	чтение/запись	RTC месяц
rtc_year	цел.	чтение/запись	RTC год

Сводный список команд

Команды

Перемещение

ld	перемещение переменной или константы в аккумулятор
ldn	перемещение дополнения переменной в аккумулятор
st	перемещение аккумулятора в переменную
stn	перемещение дополнения аккумулятора в переменную
set	настройка аккумулятора или переменной
setc	если условие истинное, настраивают переменную
res	удаление аккумулятора или переменную
resc	если условие истинное, удаляют переменную

Логические

cpl	дополнение аккумулятора или переменной
and	логический и аккумулятор с переменной или константой
andn	логический и аккумулятор с дополнением переменной или константой
or	логический или аккумулятор с переменной или константой
orn	логический или аккумулятор с дополнением переменной или константой
xor	исключающий или аккумулятор с переменной или константой
xorn	исключающий или аккумулятор с дополнением переменной или константой
shl	сдвиг влево аккумулятора, настройка LSB на нуль
shr	сдвиг вправо аккумулятора, настройка MSB на нуль
rol	вращение влево аккумулятора, копия MSB в LSB
ror	вращение вправо аккумулятора, копия LSB в MSB
fp	положительный фланг, 1, если детектируется переход снизу-вверх, 0 в другом случае
fn	отрицательный фланг, 1, если детектируется переход сверху-вниз, 0 в другом случае

Арифметические

neg	изменение знака аккумулятора
add	добавление переменной или константы в аккумулятор
sub	вычитание переменной или константы из аккумулятора
mul	умножение аккумулятора с переменной или константой
div	деление аккумулятора с переменной или константой
mod	остатки от деленного аккумулятора с переменной или константой

Сравнение

eq	тест, если аккумулятор равен значению
ne	тест, если аккумулятор не равен значению
gt	тест, если аккумулятор больше, чем аккумулятор
ge	тест, если аккумулятор больше или равна значению
lt	тест, если аккумулятор ниже, чем значение
le	тест, если аккумулятор ниже или равно знасению

Ветвление

jmp label	безусловный прыжок в положение указанное меткой
jmpc label	прыжок, если условие истинное
jmpnc label	прыжок, если условие не истинное
cal subroutine	вызов подпрограммы
calc subroutine	вызов подпрограммы, если условие истинное
calnc subroutine	вызов подпрограммы, если условие не истинное

Типовые преобразования:

x – в – y преобразование аккумулятора из типа x в тип y (бит, слово, целое число, длинное целое, реальное)

Разрешенные типовые преобразования

	бит	слово	целое	длинное целое	реальное
бит		+	+	+	
слово			+	+	
целое		+		+	+
длинное целое		+	+		+
реальное			+	+	

Разрешенные типовые комбинации

	бит	слово	целое	длинное целое	реальное	аккум.	конст.	перем.
ld	+	+	+	+	+		+	+
ldn	+							+
st	+	+	+	+	+			+
stn	+							+
set	+					+		+
setc	+							+
res	+					+		+
resc	+							+
cpl	+	+				+		+
and	+	+					+	+
andn	+	+					+	+
or	+	+					+	+
orn	+	+					+	+
xor	+	+					+	+
xorn	+	+					+	+
shl		+				+		
shr		+				+		
rol		+				+		
ror		+				+		
fp	+					+		+
fn	+					+		+
neg			+	+	+	+		
add			+	+	+		+	+
sub			+	+	+		+	+
mul			+	+	+		+	+
div			+	+	+		+	+
mod			+	+			+	+
eq	+	+	+	+	+		+	+
ne	+	+	+	+	+		+	+
gt			+	+	+		+	+
ge			+	+	+		+	+
lt			+	+	+		+	+
le			+	+	+		+	+
jmp							+	
jmpc							+	
jmpnc							+	
cal							+	
calc							+	
calnc							+	
x-to-y	+	+	+	+	+	+		
dprnx	+		+	+	+			+

Сводка структурированных текстов

Операторы

оператор	альтер.	унарный	бинарный	бит	слово	целое	дл. целое	реальное	рез-т
+			•			•	•	•	тот же самый
-		•	•			•	•	•	тот же самый
*			•			•	•	•	тот же самый
/			•			•	•	•	тот же самый
mod	%		•			•	•		тот же самый
not	!	•		•	•				тот же самый
and	&		•	•	•				тот же самый
or	|		•	•	•				тот же самый
xor	^		•	•	•				тот же самый
=	==		•	•	•	•	•	•	бит
<>	!=		•	•	•	•	•	•	бит
<			•			•	•	•	бит
<=			•			•	•	•	бит
>			•			•	•	•	бит
>=			•			•	•	•	бит
:=			•	•	•	•	•	•	тот же самый

Управление обменом данных

if…than…else (если…затем…еще)

```
if  <expression>  then
    <statements> ;
elsif  <statements>  then
    <statements> ;
else
    <statements> ;
end_if ;
```

case…of (выбор… из)

```
case  <expression>  of
    <value1> :  <statements> ;
    <value2> :  <statements> ;
    …
    <valuen> :  <statements> ;
else
    <statements> ;
end_case ;
```

for…do (для…исполнить)

```
for  <var> : =<expression>  to  <expression>  do
    <statements> ;
end_for ;
```

while…do (тогда как… исполнить)

```
while  <expression>  do
    <statements> ;
end_while ;
```

Операторы

Функции детектирования фронта (среза) (импульса)

детектирование фронта (импульса)

```
fp    (b : bit)   :bit
```

детектирование среза (импульса)

```
fn    (b : bit)   :bit
```

Функции приведения

```
int   (expression) :int;
word    (expression) :word;
long (expression) :long;
real  (expression) :real;
```

Функции дисплея

очистка дисплея

```
dclr  (slot : int);
```

печать символов ASCII

```
dprnc   (slot : int,    x : int,    y : int,    c : char);
```

печать строки

```
dprns   (slot : int,    x : int,    y : int,    str : string);
```

печать двоичного значения

```
dprnb   (slot : int,    x : int,    y : int,    c0 : char  cl : char,  value : bit);
```

печать значения целого числа

```
dprni   (slot : int,    x : int,    y : int,    width : int    zeroblank : bit,  value : int);
```

печать значения длинного целого числа

```
dprnl   (slot : int,    x : int,    y : int,    width : int    zeroblank : bit,  value : long);
```

печать реального значения

```
dprnr   (slot : int,    x : int,    y : int,    width : int    dec : bit,  value : real);
```

Условные обозначения:

slot……. номер слота
x……. положение x (0-слева)
y……. положение y (0-сверху)
width…….. печать разрядности
zeroblank…….. подавление ведущего нуля (0-нет, 1-да)
dec……. положение десятичной точки
c……. одинарный символ
str…… строка символов
value…….. значение для печати

Функции Com2

инициализации порта

```
com_init    (baud_rate : long,       data_bits : int,   parity : int,   stop_bits : int) :bit;
```

передача

```
tx    start (char    num : int) ;
tx    stop () ;
tx    count () : int ;
tx_active () : bit ;
```

получение

```
rx    start (beg ch : char,     end    ch : char, len : int,   msg    tout : int, char    tout : int) ;
rx    stop () ;
rx    count () : int ;
rx_active () : bit ;
rx_status () : int ;
```

анализ полученного сообщения

```
rx    bufrd (position : int) : int ;
rx    strcmp (position : int,  str : string) : bit ;
rx    strpos (position : int,   str : string) : int ;
rx    strtoi (position : int) : int ;
rx    strtol (position : int) : long ;
rx    strtor (position : int) : real ;
```

Функции высокоскоростного счетчика

запуск/остановка счета

```
hsc_start () ;
hsc_stop () ;
hsc_active () : bit ;
```

чтение/запись значения счетчика

```
hsc   read () : long;
hsc_write (position : long) ;
```

настройка/переустановка высокоскоростного действия

```
hsc   set     action (action : bit, variable : bit) ;
hsc   reset   action () ;
hsc_check_action () : bit ;
```

разблокировка/блокировка переустановки счетчика на входе нуля

```
hsc   enable     zero () ;
hsc   disable    zero () ;
hsc_check_zero () : bit ;
```

детектирование и считывание положения входа нуля

```
hsc   detect zero () :bit;
hsc_read_zero () : long ;
```

Специальные функции

чтение/запись текущего ip адреса

```
get   ip () : long ;
set_ip    (ip_address : long,       subnet : long,     gateway : long, dns_server : long) ;
```

чтение текущего NAD адреса (последовательный или альтернативный)

```
get_nad() : long ;
```

Набор символов

Для ввода символов, не поддерживаемых клавиатурой, нажмите кнопку Alt, на клавиатуре цифрового типа код символа упреждается 0 и отпустите Alt. Код символа должен выражаться в десятичной системе. Должна быть включена фиксация числового регистра (на клавиатуре).

Пример:

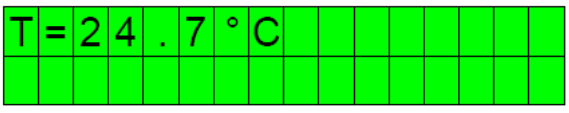
В соответствии с таблицей символов, символ "o" (градусы) имеет шестнадцатеричный код DFh. Преобразование значения в десятичное дает 223 (DFh = D0h + 0Fh = 16*13 + 15 = 223).

Ввод символа градуса:

- удостоверьтесь, что включена фиксация числового регистра (на клавиатуре)
- нажмите Alt
- нажмите последовательно 0223
- отпустите Alt

Когда идет настройка другого символа, вместо символа "o" появляется символ "β", но он будет показываться правильно.

```
dprns (1, 0, 0, 'T=xx . xβC') ;
dprnr (1, 2, 0, 4, 1, iw000*0 . 1) ;
```



Коды 0..7 резервируются для гистограммы или символов Latin-2.

Кнопочные комбинации быстрого вызова

Общепринятые

F1		Помощь
F2		Проверка синтаксиса
F5		Настройка аппаратный средств
F6		Редактор выделения
F7		Редактор маски
F8		Редактор гнезда
F9		Посылка
F10		Монитор переменной
Ctrl-F10		Менеджер данных
F11		Запуск программы контроллера (PLC)
F12		Остановка программы контроллера (PLC)
Ctrl-O		Открыть
Ctrl-S		Сохранить
Ctrl-Shift-S		Сохранить как
Ctrl-P		Печать проекта
Ctrl-D		Присоединение/отсоединение коммуникационного порта
Ctrl-L		Выбор NAD (адреса)
Ins		Вставить (в соответствии с контекстом)
Delete		Удалить (в соответствии с контекстом)
Ctrl-Up		Сдвиг позиции вверх
Ctrl-Dn		Сдвиг позиции вниз
Ctrl-Tab		Следующее окно
Ctrl-Shift-Tab		Предыдущее окно
Ctrl-F4		Закрытие окна
Alt-F4		Выход из программы

Текстовый редактор

Ctrl-space		Вставка переменной или функции
Ctrl-Z	Alt-Backspace	Отменить
Shift-Ctrl-Z		Выполнить повторно
Ctrl-X	Shift-Del	Вырезать
Ctrl-C	Ctrl-Insert	Копировать
Ctrl-V	Shift-Insert	Вставить
Ctrl-A		Выбрать все
Ctrl-F		Найти
F3		Найти следующий
Ctrl-R		Заменить
Ctrl-G		Переход на строку
Ctrl-Shift-I		Блок с отступом
Ctrl-Shift-U		Блок без отступа
Ctrl-Shift-C		Комментируемый/некомментируемый выбор