1) Основные понятия и определения систем реального времени. Применения СРВ на различных областях народного хозяйства. Понятие систем жесткого и мягкого реального времени.

СРВ — это система, правильность функционирования которой зависит от логической корректности вычислений, и от времени, за которое эти вычисления производятся. Система работает в реальном времени, если ее быстродействие обработки данных и генерирования управляющих сигналов адекватно скорости протекания физических процессов на объектах контроля или управления. Быстродействие системы реального времени больше, при большей скорости протекания процессов на объекте управления.

Принято различать системы мягкого и жесткого реального времени. **В системе жёсткого реального времени** обработка событий происходит строго за предусмотренное время, большее считается фатальной ошибкой. Примерами могут быть бортовые системы управления (на самолёте, космическом корабле), системы аварийной защиты.

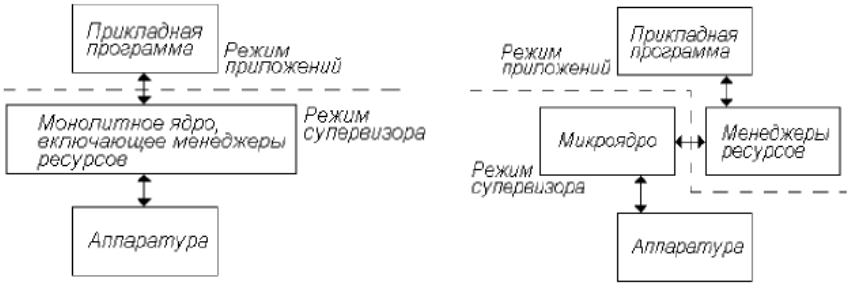
Системами мягкого реального времени называются системы, не успевающие решать задачу, но это не приводит к отказу системы в целом (все не относящиеся к жестким).

Применяются:

- **в промышленности**, включая системы управления технологическими процессами, системы промышленной автоматизации, SCADA-системы, испытательное и измерительное оборудование, робототехнику.
- **в медицине** (томографию, оборудование для радиотерапии, прикроватное мониторирование).
- **встроены в периферийных устройствах** (лазерные принтеры, сканеры, цифровые камеры, маршрутизаторы, системы для видеоконференций, мобильные телефоны, микроволновые печи, музыкальные центры, кондиционеры, системы безопасности)
- **на транспорте** (в бортовых компьютерах, системах регулирования уличного движения, управлении воздушного движения, аэрокосмической технике, системе бронирования билетов)
- **в военной технике**: системах наведения ракет, противоракетных системах, системах спутникового слежения.

2) Системы реального времени на базе операционных систем реального времени. Архитектура операционных систем реального времени. Планировщик задач. Интервал времени реакции.

В случае использования промышленных компьютеров, а также бортовых систем управления для создания программного обеспечения применяют операционные системы реального времени (ОС РВ). Примерами операционных систем реального времени являются QNX, OS-9, OS Lynx, VxWorks и т.д.(около 100).



а) Монолитная архитектура б) Микроядерная архитектура

Рис. 1.1. Типы архитектур операционных систем

Современные процессоры поддерживают два режима выполнения программного кода:

– «*режим ядра*» - имеет доступ ко всем ресурсам ЭВМ – ко внешним устройствам, к оперативной памяти по физическим адресам и пр.

– «*Режим приложений*» - работает в виртуальной среде, сформированный программами режима ядра.

Все компоненты *«монолитной»* ОС работают в режиме ядра в едином адресном пространстве. Главное достоинство таких ОС – высокая производительность. Недостатки – невозможность внесения изменений в структуру ОС(плохая масштабируемость) и невысокая реакция на внешние события (обработка этого события задерживается до возвращения на уровень приложений).

Особенность *«микроядерных»* ОС – наличие быстродействующего «микроядра», работающего в режиме ядра, а все остальные компоненты ОС работают в режиме приложений. Хорошая гибкость и масштабируемость ОС, малое время реакции на внешние события. Такие ОС отличаются относительно невысокой производительностью, т.к. при работе происходят частые переключения из режима в режим. В микроядерной архитектуре все услуги для прикладного приложения микроядро обеспечивает, отсылая сообщения соответствующим сервисам (драйверам, серверам) — другим процессам, которые, выполняются не в пространстве ядра (а в пользовательском кольце защиты).

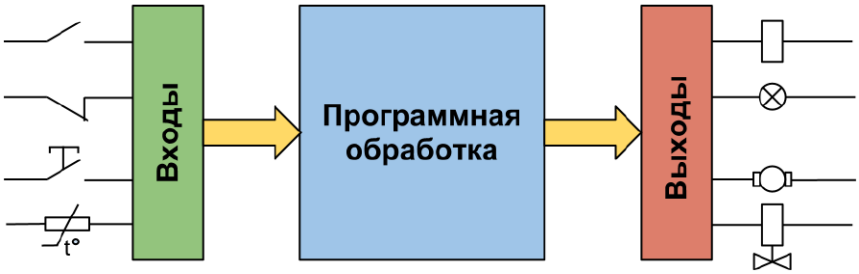
Для примера микроядро QNX всего 8 Кбайт! Дело в том, что ядро управляет только разделением времени между процессами и передачей сообщений. Даже управление процессами и распределение ресурсов для процессов осуществляется отдельным процессом, который так и называется — *менеджер процессов*.

Планировщик ОС РВ, в отличии от ОС общего назначения(которые по истечении каждого кванта времени просматривает очередь активных процессов и принимает решение, кому передать управление, основываясь на приоритетах процессов), имеют возможность сменить процесс до истечения непрерывного кванта времени ("*time slice*"), если в этом возникла необходимость. Они не могут используется *алгоритм круговой диспетчеризации* в чистом виде, из-за того, что в течение непрерывного кванта времени, процессором владеет только один процесс. ОС РВ отличаются богатством различных алгоритмов планирования, цель одна - предоставить инструмент, позволяющий в нужный момент времени исполнять именно тот процесс, который необходим.

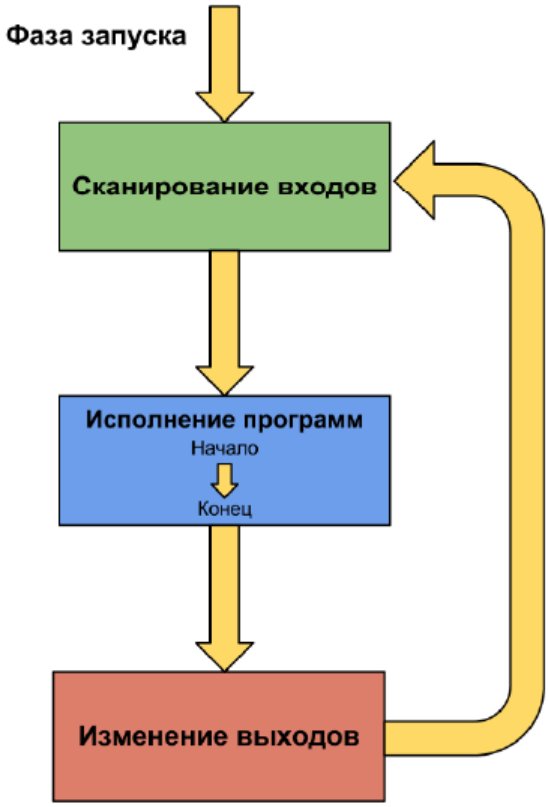
Интервал времени реакции – от возникновения запроса на прерывание и до выполнения первой инструкции обработчика определяется целиком свойствами операционной системы и архитектурой компьютера. Интервалы времени реакции для различных ОС РВ приблизительно находится в пределах 5 мкс.

3) Создание систем реального времени на базе ПЛК. Упрощенная схема работы ПЛК. Цикл контроллера. Время реакции на событие.

ПЛК *(программируемый логический контроллер)* представляют собой микропроцессорное устройство, предназначенное для сбора, преобразования, обработки, хранения информации и выработки команд управления. На входы ПЛК подключены датчики, контакты кнопок и реле, а на выходы подключены исполнительные механизмы.



ПО состоит из двух частей: *системное ПО* и *пользовательская программа*. *Системное программное обеспечение* – управляет работой узлов контроллера, отвечает за организацию связи по сети.

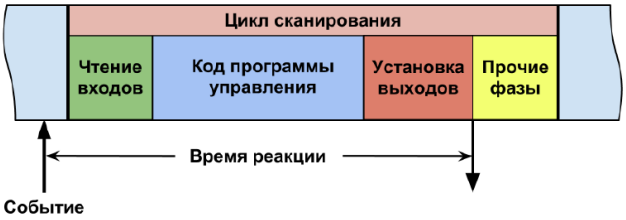


Как видно, рабочий цикл ПЛК включает 3 фазы:

1) ПЛК производит физическое *чтение входов*. Прочитанные значения размещаются в ОП контроллера в области памяти входов. Таким образом, создается полная одномоментная копия значений входов. Значения входов в процессе выполнения пользовательской программы не изменяются в пределах одного рабочего цикла.

2) *Выполняется код пользовательской программы* (работает с копиями значений входов и выходов, размещенными в оперативной памяти)

3) *Устанавливаются физические выходы* ПЛК в соответствие с расчетными значениями. Именно в этот момент происходят переключения/изменения положения исполнительных механизмов.



Время реакции на событие зависит от времени выполнения одного цикла прикладной программы. **Определение времени реакции**– от момента события до момента выдачи управляющего сигнала. Для уменьшения времени реакции в контроллерах используются циклические и аппаратные прерывания, обработчики прерываний для инициализации системы при первом запуске, для диагностирования и т.п.

Контроль времени рабочего цикла осуществляется при поддержке аппаратно реализованного «сторожевого таймера». Если фаза пользовательского кода выполняется дольше установленного порога, то ее работа будет прервана и цикл работы контроллера принудительно начнется заново. Таким образом, достигается предсказуемое поведение ПЛК при ошибках в программе и при «зависании» по причине аппаратных сбоев.

4) Обзор языков программирования промышленных контроллеров: LD, FBD, IL, ST, SFC.

В 1993 году был принят стандарт МЭК 61131-3, он устанавливает пять языков программирования ПЛК, три графических и два текстовых:

1. Релейно-контактные схемы, или релейные диаграммы (***LD*** - *Ladder Diagram*);
2. Диаграммы функциональных блоков (***FBD*** - *Function Block Diagram*);
3. Список инструкций ***IL*** - *Instruction List*).
4. Структурированный текст (***ST*** - *Structured Text*);
5. Последовательные функциональные схемы (***SFC*** - *Sequential Function Chart*);

Язык LD – язык релейно-контактных схем. Графический язык релейной логики появился в виде электрических схем, которые состояли из контактов и обмоток электромагнитных реле.

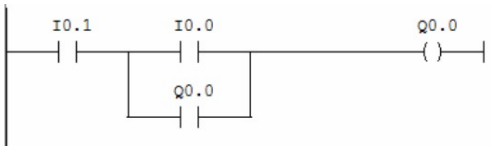


Рис. Программа включения двигателя на языке LD

Язык LD проблематично использовать для реализации сложных алгоритмов, т.к. он не поддерживает подпрограммы, функции и другие средства структурирования программ с целью повышения качества программирования. В него были добавлены функциональные блоки, которые выполняли

операции сложения, умножения, вычисления среднего и т.д. Сложные вычисления в этом языке невозможны. Недостатком является - при программировании на мониторе компьютера умещается только маленькая часть программы.

Язык FBD – Диаграммы функциональных блоков

FBD является графическим языком для программирования процессов прохождения сигналов через логические и функциональные блоки. Функциональные блоки представляют собой фрагменты программ, написанных на IL, SFC или других языках, которые могут быть многократно использованы в разных частях программы и которым соответствует графическое изображение, принятое при разработке функциональных схем электронных устройств.

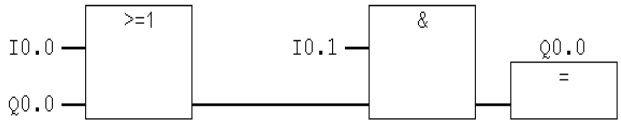


Рис. Программа включения двигателя на языке FBD

Язык IL – Список инструкций - используется для реализации функций, функциональных блоков и программ, а также шагов и переходов в языке SFC. Язык используется, когда требуется получить оптимизированный код для реализации критических секций программы, а также для решения небольших задач с малым количеством разветвлений алгоритма. На листинге 1 представлен фрагмент кода на языке IL.В основе языка лежит понятие аккумулятора и переходов по меткам. Начинается программа с загрузки в аккумулятор значения переменной. Дальнейшие шаги программы состоят в выполнении содержимым аккумулятора различных действий (их в языке всего 24).

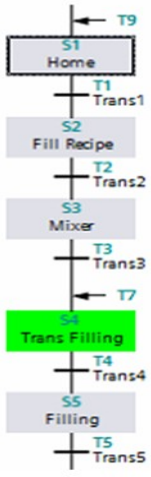
Метки	Операторы	Операнды	Комментарии
	LD	Voltage	(*Загрузить Voltage в аккумулятор*)
	GT	220	(*Если >220*)
	JMPCN	M1	(*Перейти к метке, если ">220" не верно*)
	LD	Current	(*Загрузить Current в аккумулятор*)
	SUB	10	(*Вычесть из аккумулятора 10 *)
	ST	Current	(*Присвоить Current значен. аккумулятора*)
M1:	LD	0	(*Загрузить в аккумулятор значение "0"*)
	ST	Out	(*Присвоить Out значение аккумулятора*)

Язык ST – Структурированный текст - является текстовым языком высокого и разработан специально для программирования ПЛК. Он содержит множество конструкций для присвоения значений переменным, для вызова функций и функциональных блоков, для написания выражений условных переходов, выбора операторов, для построения итерационных процессов. Этот язык предназначен в основном для выполнения сложных математических вычислений, описания сложных функций, функциональных блоков и программ.

Листинг 2. Пример программы на языке ST

```
IF Voltage>220 THEN
    (*Если V>220 В, то уменьшить ток на 10*)
   Current:=Current - 10;
ELSE
  (*Установить ток 50А и включить мотор*)
   Current:=50; Speed:= ON;
END_IF;
```

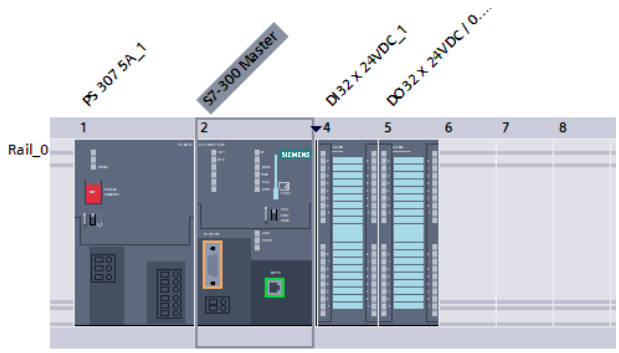
Язык SFC – Последовательные функциональные схемы - по сути это не язык, а вспомогательное средство для структурирования программ. Он предназначен специально для программирования последовательности выполнения действий системой управления, когда эти действия должны быть выполнены в заданные моменты времени или при наступлении некоторых событий. Язык SFC предназначен для описания этапов выполнения процесса.

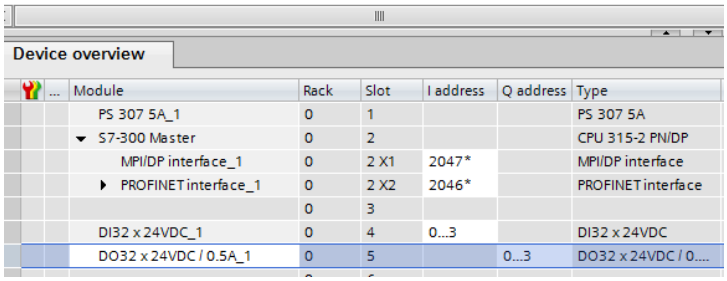


5) Создание проекта TIA Portal. Конфигурирование аппаратной части системы управления. Выбор контроллера для системы управления. Выбор источника питания и модулей ввода/вывода. Системная память контроллера S7-300.

Для создания нового проекта выполните команду **Project/New**. При этом появляется окно создания нового проекта. В этом окне введите имя проекта, при необходимости путь для сохранения проекта и нажмите на кнопку **Create**. При этом создается новый проект и появляется пустое окно проекта.

Выбор контроллера для системы управления - Для выбора управляющего контроллера выберите вид `Project view` и в дереве проектов выполните двойной щелчок на строке **`Add new device`**. В левой части окна выберите узел `Controllers`. Из этого каталога выберите контроллер модели `CPU 315-2 PN/DP` с заказным номером `6ES7 315-2EH13-0AB0`, введите имя контроллера **`S7-300 Master`** и нажмите на кнопку **`OK`**.





Выбор источника питания и модулей ввода/вывода. Откройте панель **Hardware catalog** (в правой части окна TIA Portal). Установите флажок Filter в каталоге устройств.Теперь с помощью мыши перетаскивайте источник питания **PS 307 5A** с заказным номером 6ES7 307-1EA01-0AA0 на первый слот монтажной рейки. Заметим, что при перетаскивании устройства из каталога разрешенные слоты монтажной рейки выделяются с синей границей. Далее на слот 4 перетаскивайте модуль дискретного ввода **DI32 x 24VDC / 0.5 A** с заказным номером 6ES7 321-1BH02-0AA0, а на слот 5 перетаскивайте модуль дискретного вывода **DO32 x 24VDC / 0.5 A** с заказным номером 6ES7 322-1BL00-0AA0.

Системная память CPU представляет собой входы и выходы модулей ввода вывода, а также маркеры, таймеры и счетчики. На системную память можно обращаться как биту, байту, слову и двойному слову.Для обозначения типа системной памяти используются следующие буквы:

I – входы,**Q** – выходы,**M** – маркеры,**C** – счетчики,**T** – таймеры.

6) Основы разработки программы на языке LAD. Программа нереверсивного запуска асинхронного двигателя. Создание тегов проекта. Просмотр работы программ в помощью симулятора. Просмотр кода программы в режиме мониторинга.

Программа нереверсивного запуска асинхронного двигателя

электрическая схема нереверсивного запуска асинхронного двигателя содержит следующих кнопок и исполнительных устройств:

– *кнопка Стоп;*

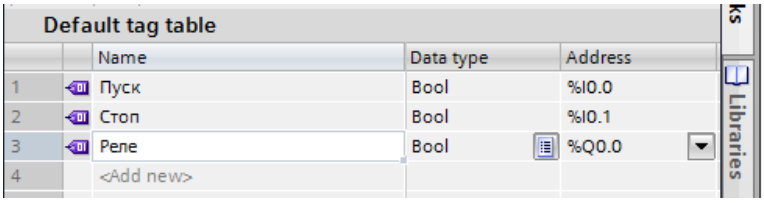
– *кнопка Пуск;*

– *реле для запуска асинхронного двигателя.*

Создание тегов проекта

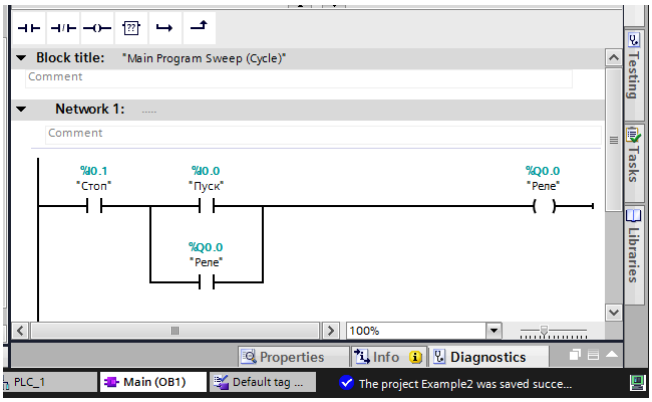
Для создания тегов → `Project view` →`Program Blocks`→**`PLC tags`**.

Как видно, для хранения имен тегов пока имеется только одна таблица: `Default tag table`. Двойным щелчком мыши откройте эту таблицу и введите имен тегов и их физических адресов, как это указано на рисунке 4.1.



Создание программного кода

`Program blocks` → `OB1` → окно кода этого блока. Теперь в окне кода организационного блока **`OB1`** введите код на языке релейно-контактных схем (**`LAD`**). Для создания разветвления после кнопки **`Стоп`** используйте кнопку , а для замыкания разветвления после кнопки **`Пуск`** используйте кнопку .



Для компиляции → **`Compile/Hardware and software`**. При этом проект компилируется и если нет ошибок, то выводит сообщение об отсутствии ошибок и предупреждений. Для быстрой отладки и просмотра результата работы программы контроллера выполните команду **`Start simulation`**. При этом открывается окно симулятора. Для загрузки PLC–проекта в симулятор выполните команду **`Download to device/Hardware and software`**. При первой загрузке проекта появляется окно, где следует выбрать интерфейс (**MPI**) и устройство, через которое программа записывается в контроллер. Теперь для загрузки проекта в симулятор нажмите на кнопку **`Load`**.

Для запуска выполнения программы в окне симулятора установите флажок **RUN-P**

1. Кнопка **`Стоп`** должна быть нормально замкнутым, поэтому вход **`I0.1`** установите на 1.

2. Для включения асинхронного двигателя нажмите на кнопку **`Пуск`** (установите вход **`I0.0`**). При этом выход **`Q0.0`** установится, т.е. реле включает асинхронный двигатель.

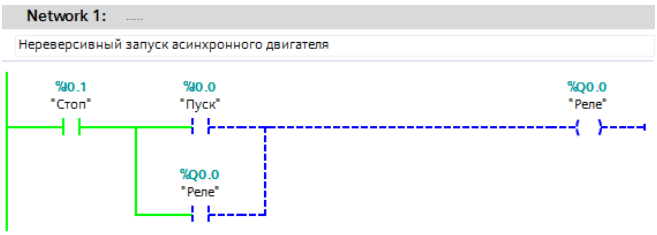
3. Теперь если даже отпустить кнопку **`Пуск`**, то выход **`Q0.0`** все равно остается включенным (благодаря, параллельно включенному контакту **`Q0.0`**).

4. Для выключения асинхронного двигателя нажмите на кнопку **`Стоп`** (**`I0.1 = 0`**). Тогда выход **`Q0.0`** обнуляется, т.е. двигатель выключается.

Просмотр кода программы в режиме мониторинга

Откройте код **`OB1`** → **`Debug/Monitor`**.

При этом активные участки программы блока `OB1` выделяются зеленым цветом.



7) Битовые логические команды:

Работают лог сигналами (1,0), находятся под узлом Bit logic operations.

---| |---: (Нормально открытый контакт) когда управляющий бит равен 1, контакт будет замыкаться (пропускать ток). (пример- нереверсивный запуск асинхронного двигателя)

---( ) : (Выходная катушка) работает как катушка реле в цепи управления релейно-контактной схемы. Выходную катушку можно установить только на правом конце логической цепи.

Если к катушке подводится ток, то управляющий бит катушки устанавливается в "1". Если к катушке не подводится ток, то управляющий бит катушки устанавливается в "0". (пример- нереверсивный запуск асинхронного двигателя)

---| / |---: (Нормально замкнутый контакт). Когда управляющий бит равен 0, то контакт замкнут (т.е. будет пропускать ток).

---(S): (Катушка установки). Эта команда устанавливает катушку с помощью короткого импульса лог. 1 на входе этой команды. Если на входе этой команды лог. 0, то эта команда не изменяет состояние катушки.

---**(R)**: (*Катушка сброса*). Эта команда обнуляет катушку с помощью короткого импульса лог. 1 на входе этой команды. Если на входе лог. 0, то эта команда не изменяет состояние катушки.

---|P|---: (Команда Выделение положительного фронта). Эта команда обнаруживает изменение сигнала на входе команды с 0 на 1 (положительный фронт) и формирует короткий импульс лог. 1 на один цикл контролера.

---| N|---: (Команда Выделение отрицательного фронта). Эта команда обнаруживает изменение сигнала на входе команды с 1 на 0 (отрицательный фронт) и формирует короткий импульс лог. 1 длиной один цикл контроллера.

Команды положительный и отрицательный фронт имеют два операнда. Первый операнд является входом для этих команд (для которого определяется фронт сигнала), в второй операнд предназначен для сохранения предыдущего состояния входного сигнала.

8) **Команды счетчиков**: Использование команд счетчика на примере промежуточного склада.

Находятся под узлом Counter operations. При создании счётчиков необходим блок данных.

Команда `CTU` (*прямой счет*). Является инкрементным счётчиком. Имеет входы: CU, RESET, PV(word) и выходы: Q(bool), CV(word). На входе CU (переход из false -> true) выход CV увеличивается на 1. Выход Q устанавливается в true, когда счётчик достигнет значения заданного в PV. Счётчик CV сбрасывается в 0 по входу RESET = true.

Команда `CTD` (*обратный счет*). Является декрементным счётчиком. Имеет входы: CD, LOAD, PV(word) и выходы: Q(bool), CV(word). На входе CV (переход из false -> true) выход CV уменьшается на 1. Когда счётчик достигнет 0, счёт останавливается, выход Q переключается в true. Счётчик CV загружается начальными значениями, равными PV по входу LOAD = true.

Команда `CTUD` (*реверсивный счетчик*). Является инкрементным/декрементным счётчиком. Имеет входы: CU, CD, Reset, Load и выходы: QU and QD(bool), PV and CV(word). По входу RESET счетчик CV сбрасывается в 0, по входу LOAD загружается значением PV. По фронту на входе CU счетчик увеличивается на 1. По фронту на входе CD счетчик уменьшается на 1 (до 0). QU устанавливается в TRUE, когда CV больше или равен PV. QD устанавливается в TRUE, когда CV равен 0.

9) **Таймерные команды**:

Находятся под узлом Timer operations.

Команда `S_PULSE` – *Таймер «Импульс»*. Запускается, если имеется положительный фронт(изменение из 0 в 1) на входе S(start). Таймер продолжает работать с заданным временем, указанным на входе TV(time value) до тех пор, пока не истечет запрограммированное время, и при условии, что состояние сигнала на входе TV=1. Пока таймер работает, состояние сигнала на выходе Q = 1. Если на входе S изменение с 1 до 0 до истечения заданного времени, таймер останавливается, тогда Q=0, а значение времени запоминается, чтобы потом вести счёт от него (если R(reset) не изменяется с 0 на 1 -> сбрасывание таймера).

Команда `S_PEXT` – *Таймер, запускаемый коротким импульсом*. Запускается, если имеется положительный фронт(изменение из 0 в 1) на входе S(start). Таймер продолжает работать с заданным временем, указанным на входе TV(time value), даже если состояние на входе S меняется на 0 до истечения времени. Пока таймер работает, Q=1. Таймер перезапускается

с заданным временем, если состояние сигнала на входе S меняется с 0 на1 во время работы таймера. Изменение с 0 на 1 на входе R во время работы таймера сбрасывает таймер + сбрасывает в ноль время и базу времени. Выходы BI и BCD используются для чтения текущего времени. На выходе BI текущее время представлено в двоичном формате, а на выходе BCD – в двоично-десятичном формате.

Команда `S_OFFDT` – *Таймер «Задержка выключения»*. Запускается, если имеется отрицательный фронт (с 1 на 0). Состояние сигнала на выходе Q = 1 тогда, когда S = 1 или если таймер работает. Если время в TV истекло, то состояние на выходе Q = 0. Если состояние сигнала на выходе S изменяется с 0 на 1 во время работы таймера, то значение времени запоминается. Если после этого S изменить с 1 на 0, то отсчёт времени будет вестись от запомненного. Изменение с 0 на 1 на входе R во время работы сбрасывает таймер, если же, когда таймер не работает, то Q = 0 и происходит сброс времени.

Использование команды таймеров для задержки выключения света в комнате.

10) **Обзор блоков проекта.**

Организационные блоки (`OB`) и прерывания. Образуют интерфейс между операционной системой CPU и программой пользователя.

OB1(блок циклического выполнения программы)	Главная циклическая программа, исполняется непрерывно. Имеет самый низкий приоритет, т.е. все остальные ob могут прерывать его выполнение. Обычно в нём организовывают последовательность выполнения операций тех процесса.
OB10–OB17(блоки прерываний по времени)	В определенное время суток или через равные промежутки времени (однократно, ежеминутно, ежечасно, в конце каждого месяца и проч). Чтобы запустить прерывание по времени, его необходимо вначале установить, а потом активировать.
OB20-OB23(блоки прерываний с задержкой)	По истечении запрограммированного времени, управляется из пользовательской программы. OB прерываний с задержкой исполняются только в том случае, когда CPU находится в режиме RUN.
OB30–OB38(блоки циклических прерываний)	Регулярно, через запрограммированные интервалы времени (например, каждые 10 мс). Нужно позаботиться о том, чтобы время работы этих блоков было значительно меньше интервала времени, через который он вызывается.Циклические прерывания удобно использовать для симуляции работы устройств управления: задвижек, дозаторов и т.д.
OB40–OB47(блоки аппаратных прерываний)	Аппаратные прерывания (по сигналу прерывания от I/O –модуля), возникновения ошибки.
OB100 –OB102	При запуске программируемого контроллера. OB100 для включения питания.

Создание и вызов функций (FC). Используется при вычислениях по мат-формулам. Создаётся в узле Program Blocks, содержит в себе раздел описания интерфейса и раздел кода программы. В первом определяются входы и выходы + временные данные функции. Во втором используют мат-функции, например на языке LAD это раздел Math functions. Вызываться могут в операционных блоках, в других функциях и функциональных блоках.

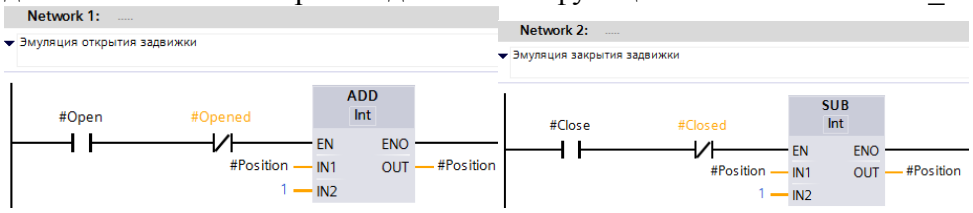
Создание и вызов функциональных блоков (FB). Создание функциональных блоков аналогично созданию функций. Единственным различием функциональных блоков от функций является то, что значений входов и выходов функционального блока можно задавать в виде блоков данных DB. Это удобно при вызове одного функционального блока для различных однотипных объектов.

Глобальные и экземплярные блоки проекта. Эти блоки хранят в себе информацию о состоянии объектов управления и доступны всем объектам программы. Однако экземплярные привязаны к функциональным блокам, поэтому они имеют такие же поля, что и входы/выходы FB.

11) **Создание функционального блока для задвижки**.

Определение входов и выходов функционального блока задвижки. Аналоговая задвижка открывается и закрывается за счет реверсивного включения асинхронного двигателя. Для открытия задвижки контроллер должен подавать команду Открыть и двигатель вращается в сторону открытия задвижки. Для закрытия задвижки контроллер должен подавать команду Закрыть и двигатель вращается в сторону закрытия задвижки. При полном открытии или закрытии срабатывают дискретные датчики положения Открыто или Закрыто. Промежуточное положение задвижки определяется аналоговым датчиком положения. Создаются параметры Open/Close (команды на открытие/закрытие задвижки), являющиеся параметрами типа Input. Датчики открытости и закрытости задвижки (Opened/Closed) определены как выходные параметров, так как полное открытие/закрытие задвижки эмулируется в коде функционального блока. Положение задвижки определяется параметром Position типа Int. Этот параметр является типа InOut, так как в программе эмуляции работы задвижки приходится считывать и устанавливать значение аналогового датчика положения.

Определение логики работы задвижки. Работу задвижки можно эмулировать за счет циклического вызова функционального блока FB_Zadv. Например, открытие и закрытие задвижки можно эмулировать за счет увеличения или уменьшения показаний аналогового датчика положения при каждом вызове функционального блока FB_Zadv.



Циклически вызов блока `FB_Zadv`. Как видно из кода FB_Zadv, если имеется команда на открытие или закрытие, то увеличивается или уменьшается значение аналогового датчика положения. Чтобы эти изменения произошли циклически с определенной частотой, функциональный блок FB_Zadv следует вызывать в обработчике циклического прерывания, например, в OB35. После необходимо привязать блок данных к функциональному блоку и в организационном блоке OB1 определить код для запуска двигателя задвижки.

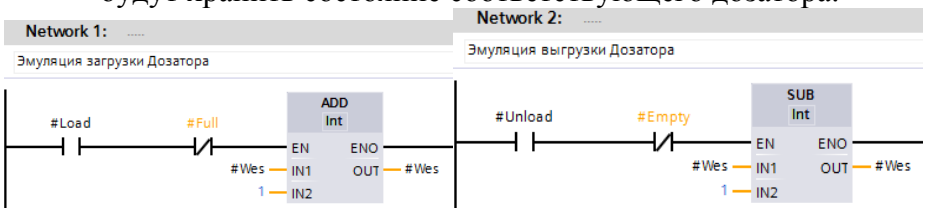
Мониторинг состояния задвижки. Состояние задвижки удобно посмотреть с помощью блока данных функционального блока (экземплярный бд). Значение поля Position, при

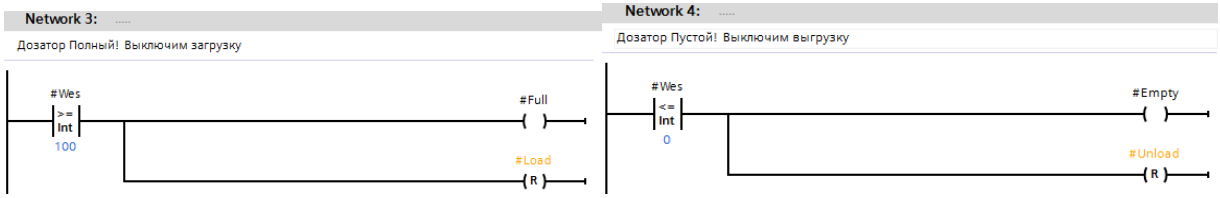
нажатии Открыть, увеличивается до 100% и процесс открытия задвижки останавливается автоматически. При Закрыть - уменьшается до 0% и процесс открытия задвижки останавливается автоматически.

12) **Использование маркерных битов для реализации последовательности выполнения операций технологического процесса (на примере бетоносмесительной установки). Просмотр работы программы управления БСУ. ХУЁВЫЙ БИЛЕТ!**

Область маркерной памяти расположена в системной области памяти процессора и доступна как для записи, так и для чтения. Элементы маркерной памяти предназначены для хранения промежуточных результатов вычислений и использование этих результатов в дальнейшем. Результат вычисления, выведенный на катушку маркерной памяти, помещается в нее сразу и становится доступным для считывания уже в следующей строке. В релейной логике STEP7 биты маркерной памяти используются одновременно для вывода результата на катушки реле и для ввода значений с контактов этих катушек.
Основное назначение битов маркерной памяти:
1)сохранять промежуточный результат при разбиении длинных релейных цепочек или в некоторых команд;
2)организовывать обратные связи или проверки условий в тактируемых схемах (в многотактных схемах).
Разбиение релейной схемы на ряд более коротких схем улучшает читабельность схемы и её лучшее понимание. Возможность реализации обратных связей позволяет реализовывать релейные схемы включения оборудования в строго определенной последовательности
Крч 18 задвижек (см.билет 11), 6 дозаторов, 2 смесителя.
Дозатор для набора определённого количества материала (песок, щебень и проч). Для работы нужно открыть одну из входных задвижек, проследить за состоянием весом, закрыть задвижку. Затем песок+щебень+цемент+минеральный порошок ->открыть выходную задвижку и высыпать всё в смеситель.

1. Создаём FB (FB_Dozat) из узла Program Blocks на языке LAD.
2. Определяем входы/выходы блока. Входы Load/UnLoad - команды загрузки и выгрузки дозатора. Выходы Full и Empty - дискретные датчики уровня (дозатор Полный и Пустой). Датчики уровня приходится определить как выходы, ибо их состояние приходится эмулировать программно. Параметр Wes типа InOut представляет собой датчик веса. Состояние веса также приходится эмулировать в коде функционального блока.
3. Если дана команда на загрузку дозатора значение датчика веса будет увеличиваться, за счет циклического вызова функционального блока FB_Dozat. Если дана команда на выгрузку дозатора - будет уменьшаться. Для каждого дозатора определите экземплярный блок данных DB_Dozat. Эти блоки данных будут хранить состояние соответствующего дозатора.





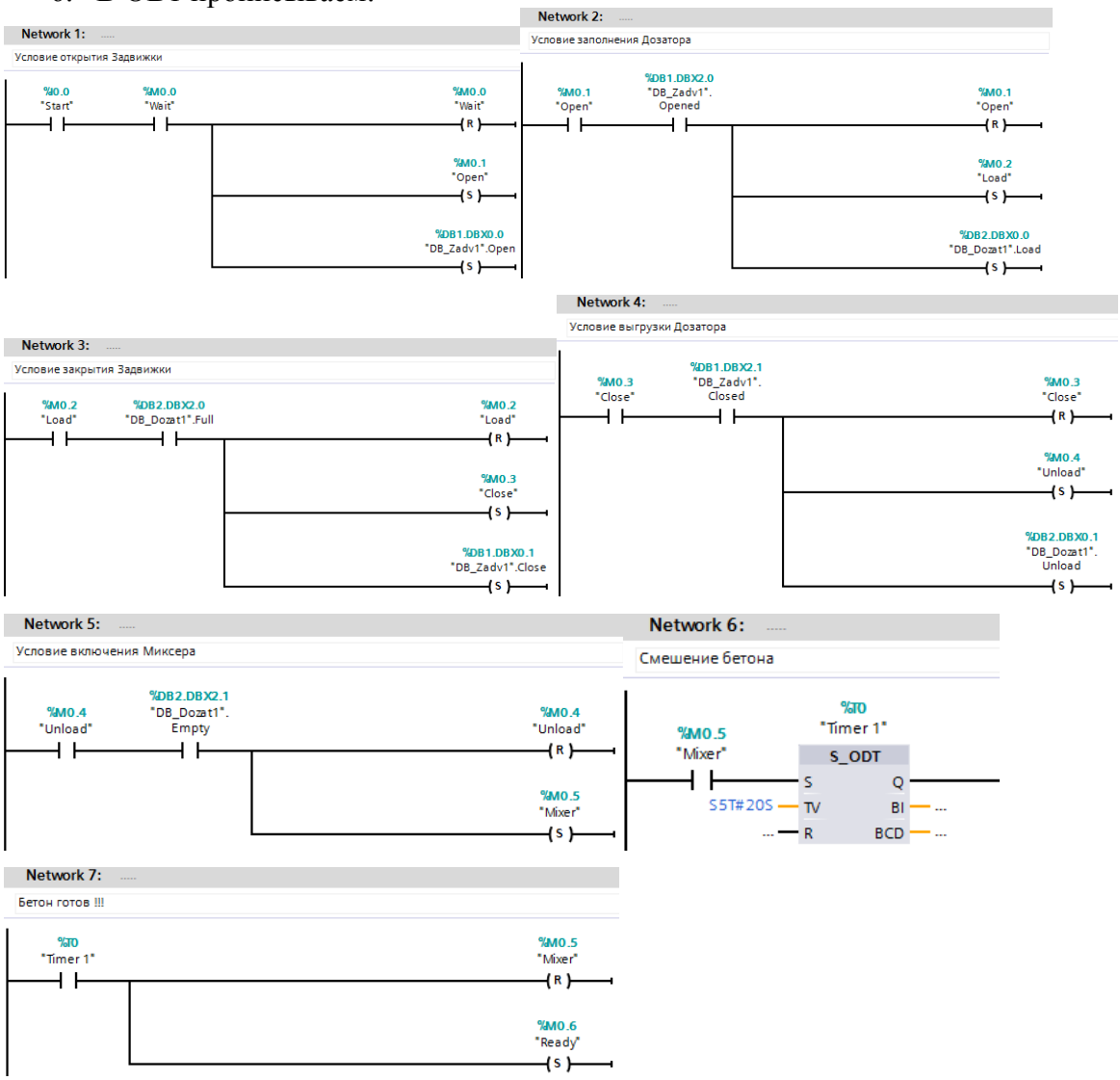
4. Вызываем блок в OB35.
5. МАРКЕРНЫЕ БИТЫ. Создаём теги в таблице Markers, которую тоже создаём в узле PLC tags. Адреса определены так, чтобы теги включались друг за другом.

Markers

	Name	Data type	Address
	Start	Bool	%I0.0
	Wait	Bool	%M0.0
	Open	Bool	%M0.1
	Load	Bool	%M0.2
	Close	Bool	%M0.3
	Unload	Bool	%M0.4
	Mixer	Bool	%M0.5
	Ready	Bool	%M0.6

Они для реализации этапов приготовления бетона.

6. В OB1 прописываем:



7. Запускаем симулятор в режиме выполнения программы. В блоках данных задвижки и дозатора вкл мониторинг. Нажимаем Пуск. ОДНАКО! Для удобства создаём таблицу для просмотра из узла Watch and force tables, в которую добавляем поля датчиков задвижки и дозатора (position, wes). И вот, моня мониторить норм.

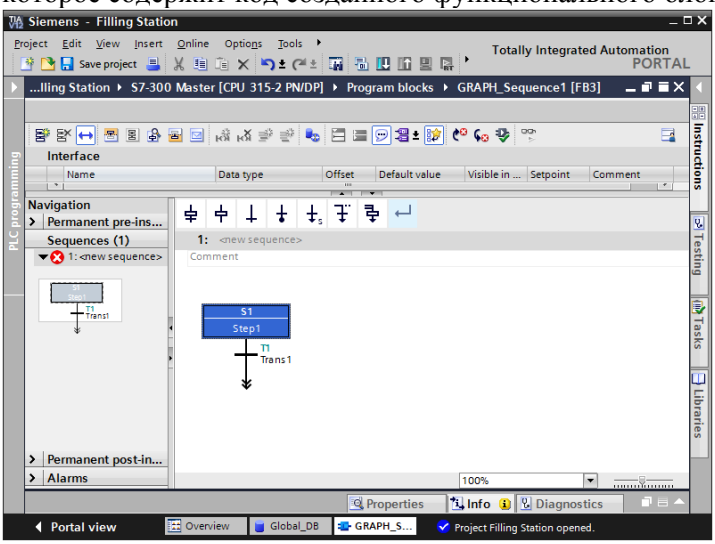
13) Использование редактора Graph для реализации программы управления процессом приготовления и розлива сока.

Графический редактор **Graph** позволяет программировать программу управления на языке **SFC** в виде последовательности действий. Для создания нового функционального блока выполните следующие действия:

– Откройте узел **Program blocks**.

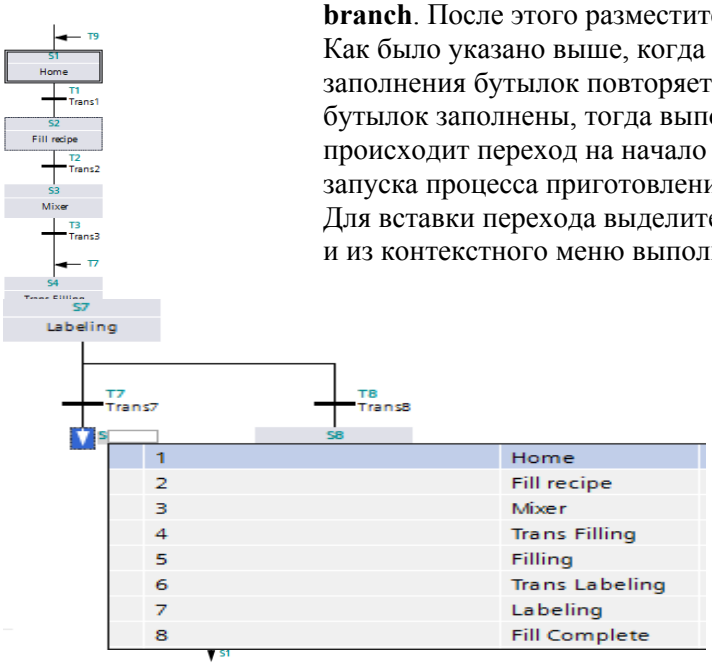
– Выполните двойной щелчок на строке **Add new block**.

При этом появляется окно создания блока проекта (рис. 7.3). В этом окне выберите тип блока Function block, введите имя блока **GRAPH_Sequence** и выберите язык программирования GRAPH. После нажатия кнопки OK, откроется в рабочей области откроется окно редактора GRAPH, которое содержит код созданного функционального блока.



Пока функциональный блок GRAPH_Sequence содержит только один шаг. Теперь добавьте шаги и переходы для процесса приготовления и розлива сока. Для этого выделите двойную стрелку шага и из контекстного меню выполните команду **Insert element/Step and transition**.

Как видно, после шага 7 определено разветвление потока выполнения. Для создания разветвления выделите шаг S7 и из контекстного меню выполните команду **Insert element/Open Alternative branch**. После этого разместите шаг S8 – Fill Complete.



Как было указано выше, когда заполнено менее 10 бутылок, процесс заполнения бутылок повторяется с шага **S4 – Trans Filling**. Когда 10 бутылок заполнены, тогда выполняется шаг **S8 – Fill Complete** и далее происходит переход на начало последовательности (шаг **S1 Home**) для запуска процесса приготовления сока по новой.

Для вставки перехода выделите конец стрелки после шага **S7 – Labeling** и из контекстного меню выполните команду **Insert element/Jump**. При этом появляется список шагов.

Выберите из этого списка шаг перехода **S4 – Trans Filling**. При этом двойная стрелка преобразуется на одинарную стрелку и рядом со стрелкой появляется надпись S4. Также на верху шага S4 появляется стрелка с надписью **T7**. Аналогично размещайте другой переход: после шага **S8** выполните переход на шаг **S1 – Home**.

14) Программирование действий шагов в редакторе Graph: квалификаторы действий N, R, S, L и события S0 и S1

Стандартные действия

Стандартные действия приведены в таблице 4.1.

Таблица 4.1

Команда	Идентификатор	Адрес	Значение
N[C]	Q, I, M, D	m,n	Адрес однократно получает значение 1
S[C]	Q, I, M, D	m,n	Адрес устанавливается в 1
R[C]	Q, I, M, D	m,n	Адрес сбрасывается в 0
D[C]	Q, I, M, D	m,n	Задержка включения на *n* секунд
L[C]	Q, I, M, D	m,n	Ограничение продолжительности адреса на *n* секунд
CALL[C]	FB, FC, SFB, SFC	Номер блока	Вызов блока

Все стандартные действия (команды) могут комбинироваться с условием блокирования (добавляется символ C). Такие действия выполняются только тогда, когда блокирование снято.

Действия, которые зависят от событий

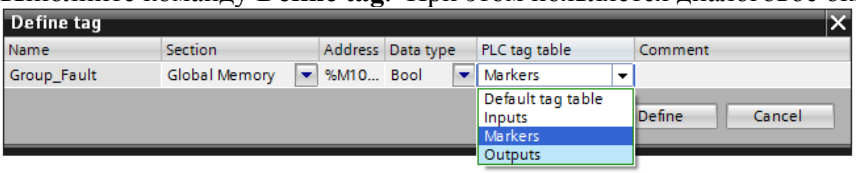
Действия могут быть логически скомбинированы с событиями. Событие – это изменение состояния шага или супервизора, блокирование, квитирование, сообщение или установка регистрации.

S1: Шаг активируется.

S0: Шаг деактивируется.

15) Программирование условий перехода на следующий шаг на примере индивидуального задания по СРВ. Создание разветвлений в программе Graph

Откройте в редакторе программ шаг **S1-Home**. Разместите нормально замкнутый контакт в области T1–Trans1. Введите имя тега **Group_Fault** – тег, который предназначен для хранения групповой ошибки. Пока этот тег не определен. Для определения этого тега из контекстного меню выполните команду **Define tag**. При этом появляется диалоговое окно создания тега



В этом окне определите тег со следующими свойствами:

– Адрес: **M10.0**

– Тип данных: Bool

– Таблица тегов: Markers

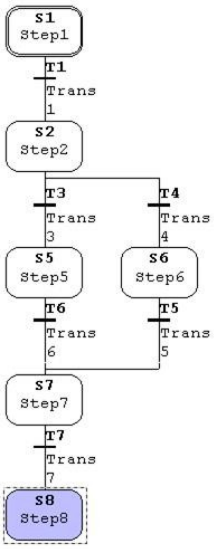
Подтвердите создание тега путем нажатия кнопки **Define**.

Далее скопируйте этот нормально замкнутый контакт, откройте каждый шаг проекта и скопированный замкнутый контакт вставьте в область перехода каждого шага.

Параллельное разветвление отвечает обработке ветвей по логике И. Параллельное разветвление начинается *общим переходом*, который активирует первый шаг *во всех* параллельных ветвях.

На рисунке 4.4 общими переходами являются переходы T3 и T7.

Параллельное разветвление заканчивается шагом, подключенным к *общему* финальному переходу. Финальный переход позволяет следующий шаг только тогда, когда выполнены все параллельные ветви.



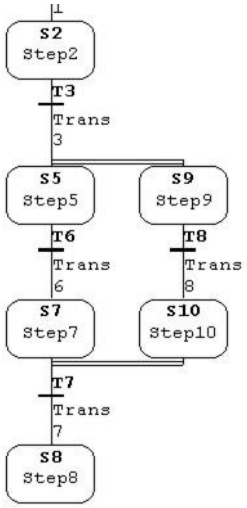


Рисунок 4.3 - Секвенсор с альтернативными ветвями

Рисунок 4.4 – Секвенсор с параллельными ветвями

Постоянные условия

Условия, которые нужно выполнять в более чем одной точке секвенсора, могут быть запрограммированы как постоянные условия.

Для программирования условий можно использовать элементы контактной схемы – нормально разомкнутые или нормально замкнутые контакты, компараторы или элементы FBD. В постоянном условии можно использовать до 32 элементов.

Результат вычисления условий хранится элементом «катушка» в LAD или блоком памяти в FBD.

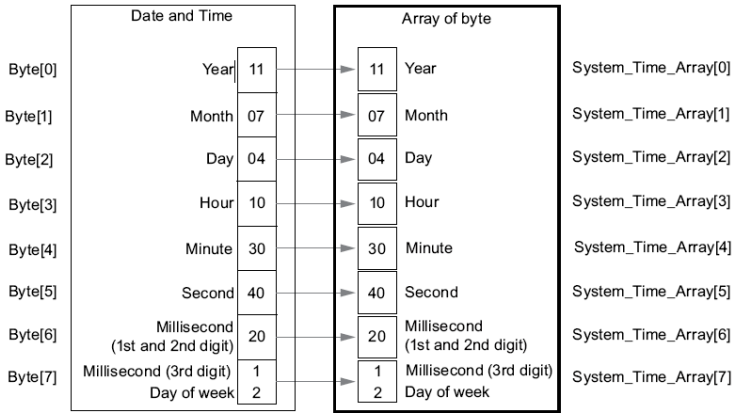
Вызовы блока

Блоки, созданные на других языках программирования, могут быть вызванны с использованием постоянных инструкций или действий в FB S7-GRAPH. После того, как вызванный блок будет выполнен, выполнение FB S7-GRAPH будет продлено.

В S7-GRAPH можно вызвать функции (FC) и функциональные блоки (FB), запрограммированные на LAD, FBD, STL или SCL, а также системные функции (SFC) и системные функциональные блоки (SFB). Функциональным блокам и системным функциональным блокам должны быть назначены экземплярные блоки данных DB. Имена блоков можно использовать в абсолютном виде, например, FC1 или символьно, например, Motor1. При вызове блоков нужно обеспечить формальные параметры вызываемого блока действительными значениями.

16) Программа вычисления срока годности сока на языке SCL (на примере технологического процесса приготовления сока). Поля типа данных Date_And_Time. Перекрытие тега типа Date_And_Time с массивом байтов/

Тип данных **`Date_And_Time`** содержит дату и время в `BCD` формате в памяти длиной 8 байт. Например, ниже указано хранение значения `2011-07-04-10:30:40.201 (July 4, 2011, 10:30, 40 seconds 201 milliseconds`). Последние 4 бита байта 7 хранят день недели.



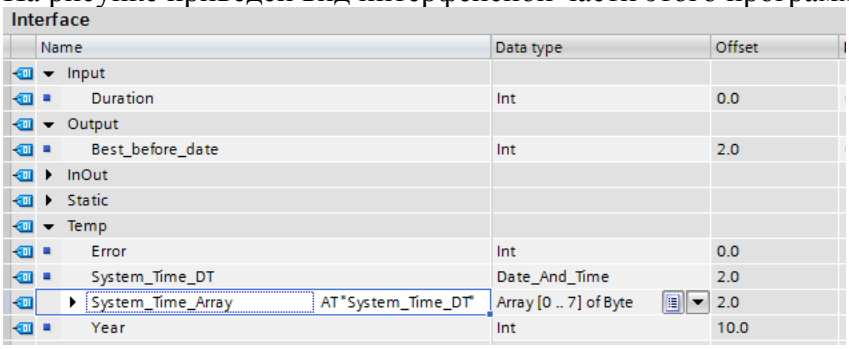
Перекрытие тега типа Date_And_Time с массивом байтов

1. Для перекрытия предыдущего тега с другим тегом, в разделе **Temp** определите третий параметр со следующими свойствами:

– Имя: System_Time_Array
– Тип данных: AT
После ввода ключевого слова **AT** автоматический создается составное имя для нового тега, которое имеет следующий формат:
<Имя нового тега> AT <Имя первого тега>
Тип данных нового тега автоматически определяется как Date_And_Time. Измените этот тип на тип **Array [0 .. 7] of Byte**.

2. В разделе **Temp** определите 4-й параметр со следующими свойствами:

– Имя: **Year**
– Тип данных: Int
Этот параметр предназначен для хранения года, прочитанного из системного времени.
На рисунке приведен вид интерфейсной части этого программного блока.



Интерфейсная часть блока SCL_Best_before_date

3. Вычисление срока годности. Для определения кода функционального блока на языке SCL переходите в область кода и наберите следующий код:

```
#Error := "RD_SYS_T"  (OUT => #System_Time_DT);
#Year := BCD_TO_INT(#System_Time_Array[0]);
#Best_Before_Date := #Year + 2000 + #Duration;
```

Как видно, в коде с помощью функции RD_SYS_T считывается системное время и из него выделяется год. Затем считывается срок годности, путем сложения числа 2000 и значения переменной #Duration.

17) Программа управления конвейером на языке STL. Система команд языка STL и разъяснение этих команд по аналогии с командами языка LAD.

Для создания STL функции STL-Conveyor выполните следующие действия:
1. В дереве проекта откройте папку Program blocks и выполните двойной щелчок на строке **Add new block**.

При этом появляется окно создания блока. В этом окне определите следующих свойств блока:
– Выберите тип блока Function
– Введите имя блока **STL-Conveyor**
– Выберите язык STL
После нажатия кнопки OK создается функция STL-Conveyor.
2. В интерфейсной части этой функции определите следующих параметров:
В разделе **Input** определите входные параметры со следующими свойствами:
Этот параметр будет использован для включения ленточного конвейера (Имя: Start_Input, Тип данных: Bool);
Этот параметр будет использован для определения направления движения конвейера (Имя: Direction, Тип данных: Bool);
В разделе **Output** определите выходные параметры со следующими свойствами:
Этот параметр будет использован, чтобы определить конвейер включен или нет (Имя: Conveyor_Done, Тип данных: Bool);
Этот параметр будет использован для управления движением конвейера Вперед (Имя: **Forward**, Тип данных: Bool);
Этот параметр будет использован для управления движением конвейера Назад (Имя: **Backward**, Тип данных: Bool).
3. Программирование управлением конвейера
Программная часть этой функции будет содержать три сегмента (network):
– В первом сегменте вы будете запрашивать, конвейер должен включаться Вперед. Если это так, то должен устанавливаться выход Forward на 1. Также выход Conveyor_Done должен быть обнулен, чтобы показать, что конвейер включен.
– Во втором сегменте вы будете запрашивать, конвейер должен включаться Назад. Для этого вы должны запрашивать входной параметр Start_Input установлен и другой входной параметр Direction не установлен. Если это так, то должен устанавливаться выход Backward на 1. Также выход Conveyor_Done должен быть обнулен, чтобы показать, что конвейер включен.
– В третьем сегменте вы запрашиваете, входной параметр Start_Input не установлен. В этом случае, оба выхода конвейера должны обнуляться и выход Conveyor_Done должен быть обнулен.
Для реализации этого выполните следующие действия:
1. В сегменте 1 определите следующий код 1:

```
A  #Direction
A  #Start_Input
S  #Forward
R  #Conveyor_Done
```

2. Вставьте второй сегмент, выбирая из контекстного меню команду Insert network. Во втором сегменте определите следующий код:

```
AN #Direction
A #Start_Input
S #Backward
R #Conveyor_Done
```

3. Вставьте третий сегмент и в нем определите следующий код:

```
AN #Start_Input
R #Forward
R #Backward
S #Conveyor_Done
```

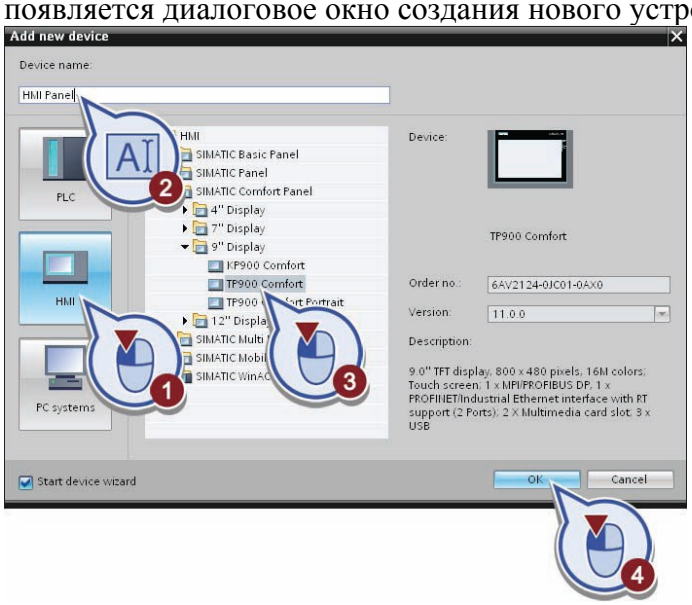
18) Создание HMI-проекта на примере панели TP900 Comfort: установление связи между контроллером и HMI-устройством; отображение вспомогательных окон; использование шаблонов экрана процесса.

Конфигурирование HMI–панели TP900 Comfort

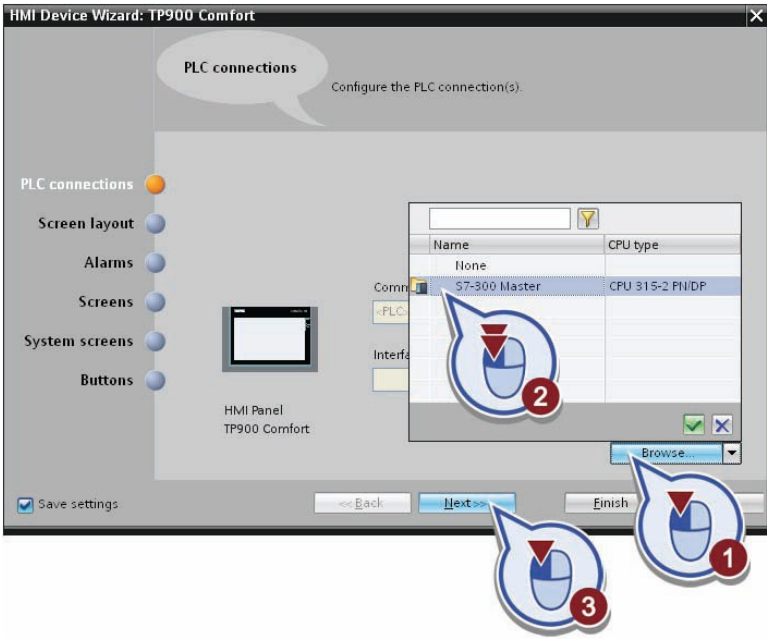
1. Создание HMI проекта

Для создания HMI проекта выполните следующие действия:

1. В дереве проектов выполните двойной щелчок на строке **Add new device.** При этом появляется диалоговое окно создания нового устройства:



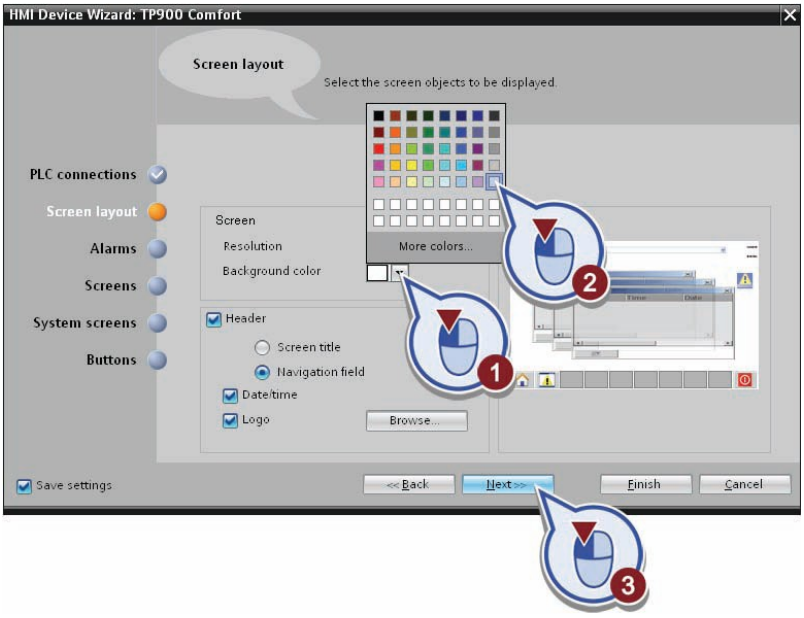
2. В этом окне выберите устройство **TP900 Comfort** и установите флажок Start device wizard. После этого нажмите на кнопку **OK**, при этом открывается диалоговое окно HMI Device Wizard:



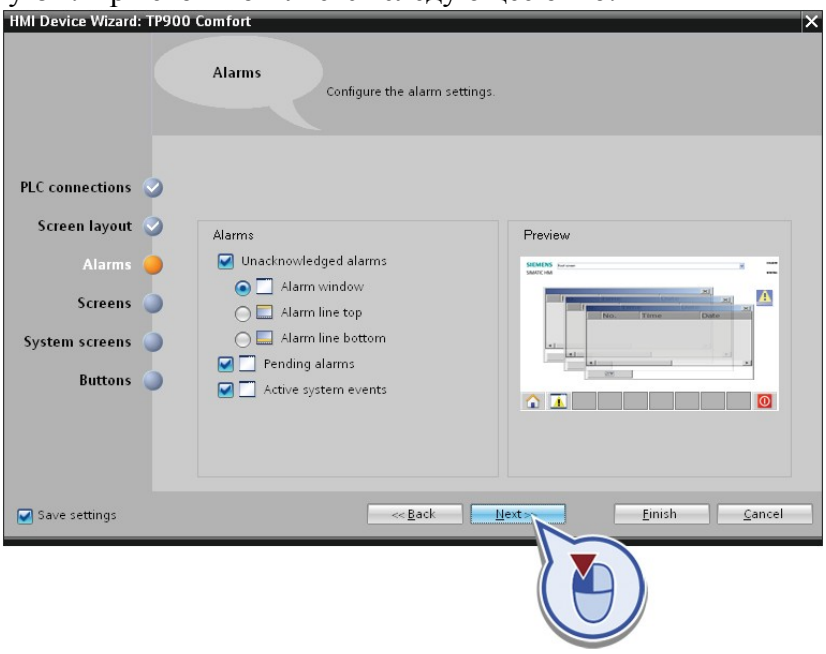
3. В этом окне определите соединение HMI-панели с контроллером:

– Щелкните на кнопку **Browse** и выберите из этого списка контроллер **S7-300 Master.**

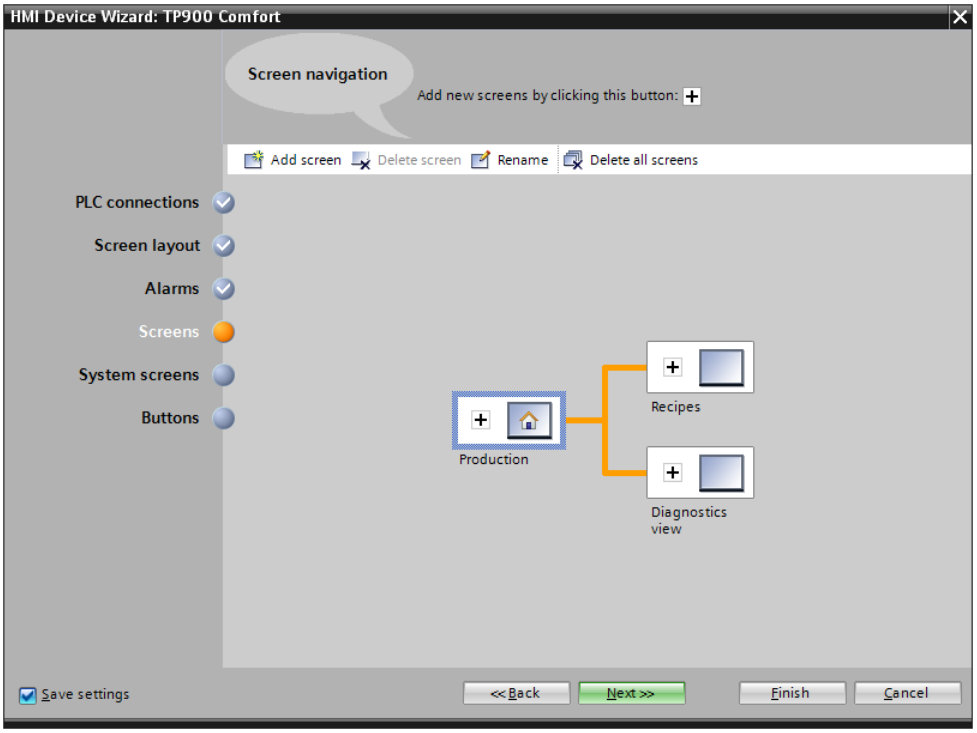
– Нажмите на кнопку **Next**, при этом появляется следующее диалоговое окно:



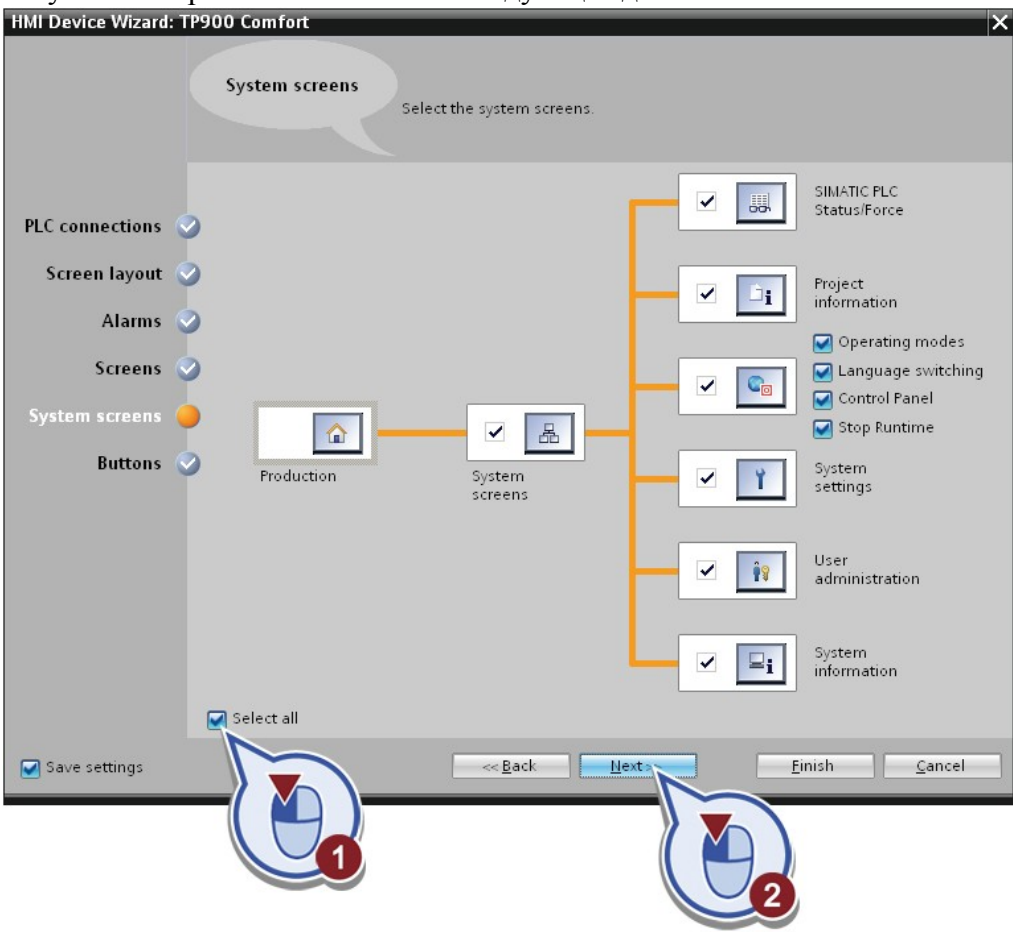
4. В этом окне выберите фоновый цвет HMI–экрана (например, белый цвет) и нажмите на кнопку **OK**. При этом появляется следующее окно:



5. Убедитесь, что в этом окне установлены соответствующие флажки, как это указано выше и нажмите на кнопку **Next.** При этом появляется следующее окно:

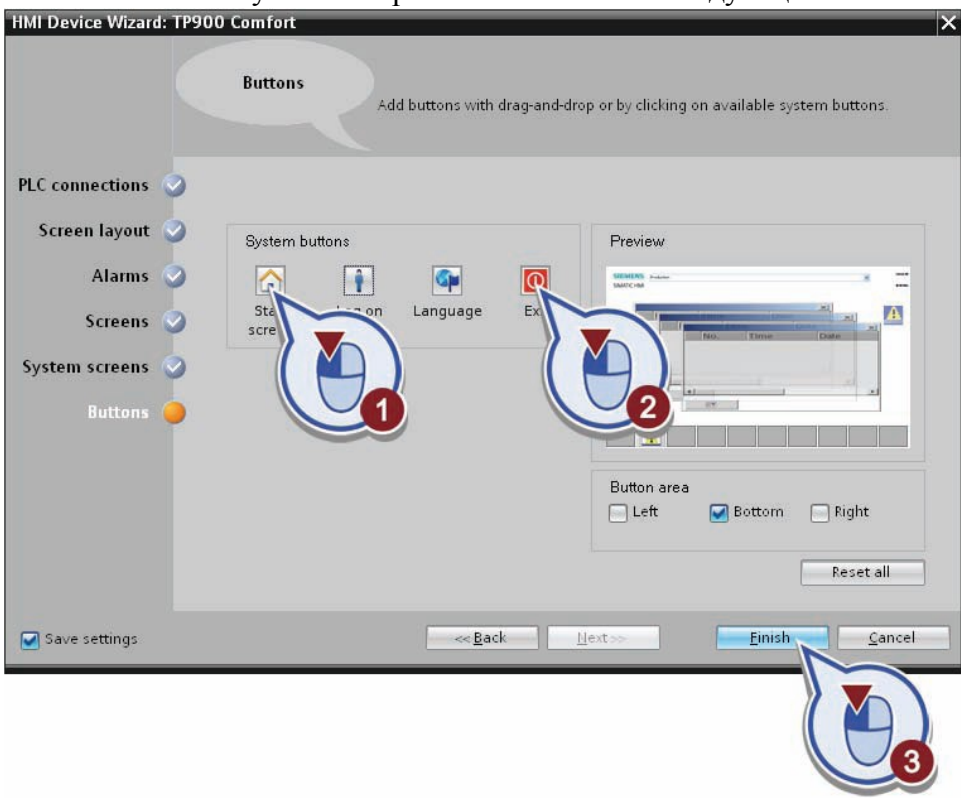


6. При необходимости добавьте новые экраны с помощью кнопки **`Add screen`**. Если эта кнопка не активна, то выберите предыдущий экран.
Переименуйте экраны как: `Production`, `Recipes` и `Diagnostics view` и нажмите на кнопку **`Next`**. При этом появляется следующее диалоговое окно:

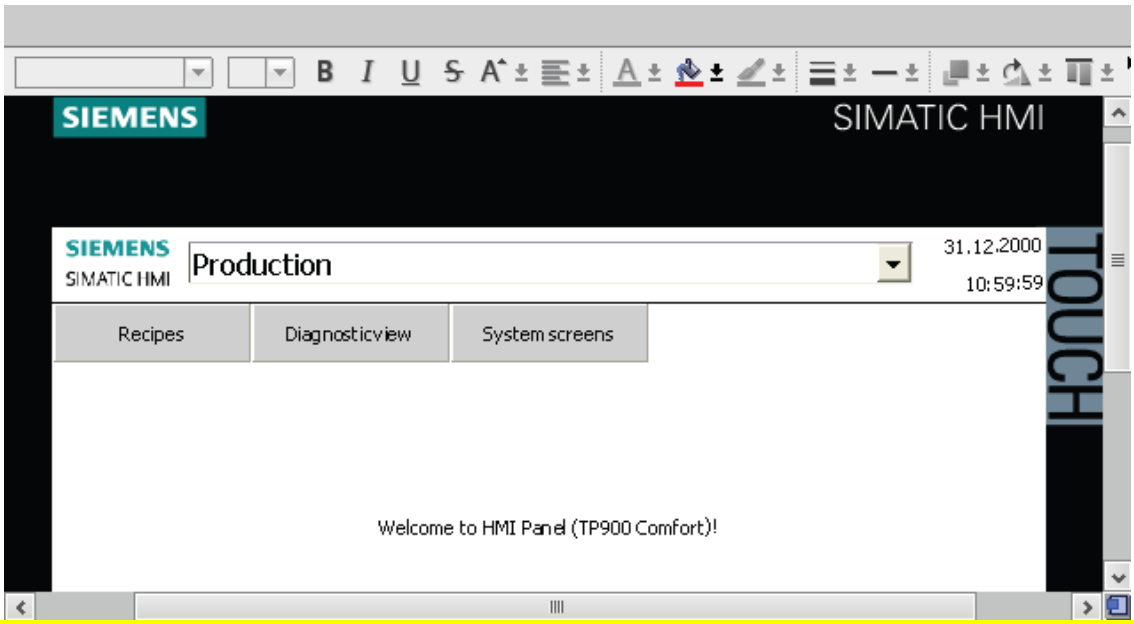


Вы можете использовать системные экраны для настройки проекта, а также для администрирования HMI-экранов. Кнопки для перехода между главным экраном Production и системными экранами создаются автоматически.

7. В этом окне во вкладке System screens установите флажок Select all и нажмите на кнопку **Next.** При этом появляется следующее окно:

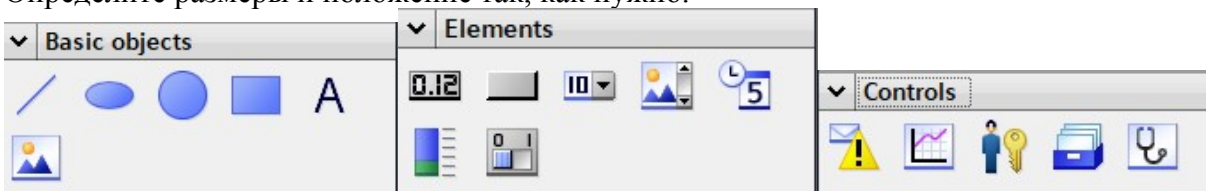


8. В этом окне добавьте кнопок Start screen и Exit и закройте Мастера конфигурирования HMI–панели путем нажатия на кнопку **Finish.** В результате был создан шаблон для HMI–экранов и автоматически открывается главный экран **Production**:

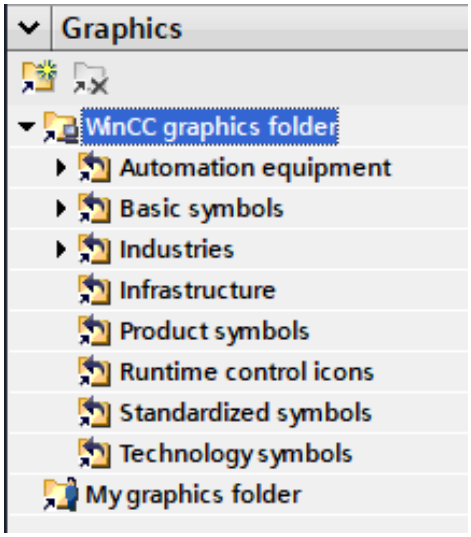


19) **Создание главного экрана процесса.** Панель инструментов графических объектов: Basic objects, Elements, Controls, Graphics.

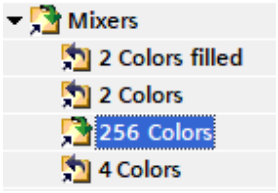
Главный экран является начальным экраном, который загружается сразу после запуска программного обеспечения HMI–экрана. В папке Screens этот экран отмечается зеленой стрелкой.
Добавление элемента на экран на примере ленточного конвейера:
1. Вначале удалите текст Welcome to.... Для этого выделите этот текст и из контекстного меню выберите команду **Delete**.
2. В панели инструментов Toolbox откройте палитру рисунков Graphics. В этой палитре рисунков содержатся различные графические объекты, которые сортированы по тематике.
3. На экран добавьте рисунок ленточного конвейера:
- Откройте каталог **WinCC graphics folder**>Automation equipment> **Conveyors, miscellaneous**.
– Выберите папку 256 Colors.
– Перетаскивайте рисунок Horizontal conveyor with perspective на главный экран. Определите размеры и положение так, как нужно.



Controls: 1)Alarm views; 2)Trend views; 3)User v.; 4)Recipe v.; 5)System diagnostics v.



Automation equipment: blowers (вентиляторы), conveyors, mixers, motors, pipes (трубы), pumps (насосы), scales (весы), tanks (ёмкости), valves (задвижки) и тд.
Basic symbols: arrows (стрелки), flags.
Industries: chemical, food, laboratory, mining (добыча), power (производительность), pulp&paper (бумага), water&wastewater (вода) и тд.
Infrastructure: buildings, containers, nature и тд.

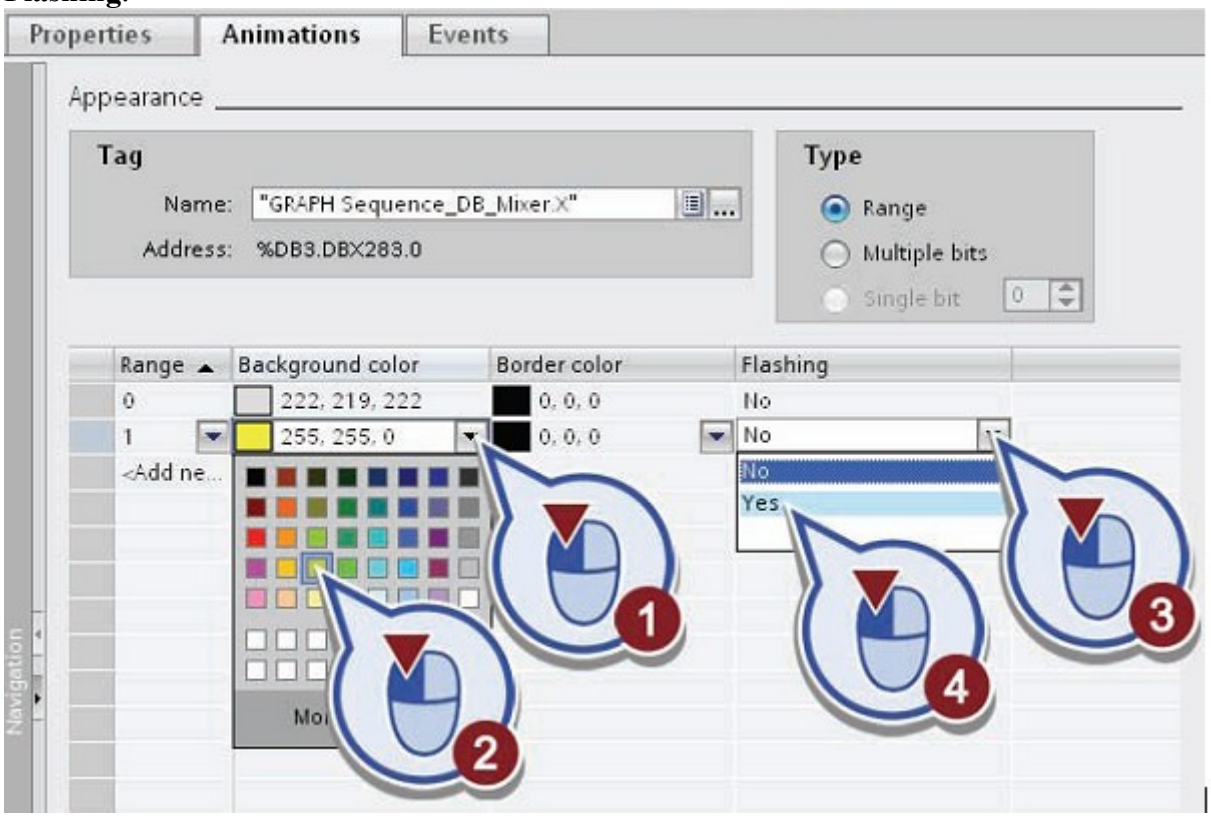


20) **Анимация графических объектов экрана процесса.** Теги состояния для шагов и переходов программы Graph: *X, S1, S0.* Изменение цвета и мерцание, отображение и исчезновение графического объекта.
Для каждого шага и перехода в программе на языке Graph создаются теги состояния. Эти теги состояния представляют собой структурный тип.
Ниже приведено описание основных полей структурного типа **для шага**:

Имя поля	***Описание поля***	***Тип поля***
X	*Step is active*	*BOOL*
S1	*Step is activated*	*BOOL*
S0	*Step is deactivated*	*BOOL*
T	*Total step activation time*	*TIME*

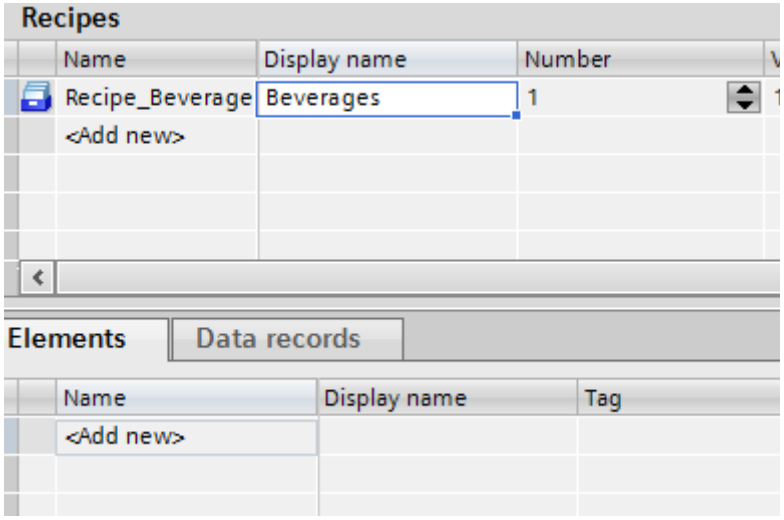
Некоторые теги можно рассматривать также как событий. Например, тег S1 состояния шага устанавливается, когда этот шаг активизируется (событие активизирования шага). Теги состояния шага и переходов размещены в блоке данных, который создается при вызове функционального блока, написанного на языке Graph. Например, GRAPH_Sequence_DB.
Изменение цвета и мерцание на примере анимации работы смесителя:

– Выберите рисунок смесителя и в окне свойств выберите вкладку **Animations**.
– Для добавления динамики мигания в узле Display щелкните на строке **Add new animation** и выберите тип динамизации **Dynamize colors and flashing**.
– Для анимации в поле имени тега введите имя тега. Для этого рядом с полем имени тега щелкните на кнопку, при этом открывается список тегов. В этом списке вначале выберите раздел **S7-300 Master>Program blocks>GRAPH Sequence DB**, а далее выберите шаг **Mixer**. При этом в правой части окна появляются теги состояния шага Mixer, выберите из этого списка тег **"X"**.
– В области **Range** определите пределы изменения тега "0" и "1".
- Выберите строку "1", определите другой фоновый цвет и включайте функцию анимации **Flashing**.



Отображение и исчезновение граф. объекта на примере бутылок:
– Во вкладке Animations откройте узел **Display** и щелкните на строке Add new animation. Далее выберите тип анимации **Make visibility dynamic**.
– В поле Tag введите имя тега для анимации. Для этого щелкните на кнопке рядом с этим полем, при этом открывается окно выбора тегов. В этом откройте блок данных S7-300 Master>Program blocks> GRAPH Sequence DB и выберите раздел Trans Filling. Далее в правой части окна выберите тег "X".
В качестве пределов изменения тега выберите от "0" до "0". В области Visibility выберите **Invisible.**
21) **Анимация графических объектов экрана процесса.** Перемещение по горизонтали и по вертикали. Реализация сложных видов анимации, путем составления составных графических объектов.
Перемещение по горизонтали и по вертикали на примере перемещения бутылок на конвейере:
Для создания анимации движения выполните следующие действия:
– В окне свойств откройте вкладку **Animations**

– В разделе Movements щелкните на строке **Add new animation**, и далее выберите вид движения Move object horizontally. При этом открывается диалоговое окно "Horizontal movement".
Для реализации анимацию движения в поле Tag следует создать тег. Для добавления тега в таблицу **Default tag table** щелкните на кнопке **Add object**. При этом открывается окно создания тега. В этом окне введите имя тега **Position_Bottle**. Из раскрывающегося списка Data type выберите тип тега Int и нажмите на кнопку **ОК**.
- Теперь в окне свойств горизонтального перемещения бутылки, в качестве верхнего значения изменения тега введите число "50".
Сложные виды анимации на примере:
Для реализации движения необходимо симулировать значений тегов Position Bottle и Position Bottle(1). Для симуляции тегов выполните следующие действия:
1. Установите курсор мыши на свободное место главного экрана и откройте окно свойств, а далее откройте вкладку Events
2. Открывайте раскрывающиеся список **<Add function>** и из раздела System functions>Other functions выберите функцию SimulateTag.
3. Соедините функцию SimulateTag с тегом Position_Bottle. Для этого откройте список тегов и из таблицы тегов Default tag table (под узлом HMI Panel>HMI Tags) выберите тег Position_Bottle.
4. Измените максимальное значение тега Position_Bottle на "50" и измените увеличение значения тега на "2" в каждом цикле:
5. Создать новую такую же функцию (можно скопировать эту), только используя тег Position_Bottle(1).
Обнуление значений симуляционных переменных
Когда выполняются шаги S4 Trans Filling и S6 Trans Labeling, бутылки должны вернуться на исходное положение. Для реализации этого следует установить соответствующих тегов на "0".
1. Для обнуления этих тегов при входе на соответствующий шаг создайте событий для тегов состояния этого шага. Для этого выполните следующие действия:
– Откройте таблицу тегов Default tag table, которая находится под узлом HMI Panel>HMI Tags.
– Выберите тег **GRAPH Sequence_DB_Trans Filling.X** и откройте вкладку Events.
– В разделе **Calculation script**, выберите функцию **SetTag**:
– Закрепите на эту функции тега Position_Bottle из таблицы тегов Default tag table.
Использование **элемента Bar** для показа заполненных бутылок:
Для показания его будем использовать столбчатую гистограмму:
1. Для этого откройте палитру объектов **Elements** в панели инструментов. Разместите объекта **Bar** на правой части главного экрана:
2. Откройте окна свойств гистограммы и введите максимального значения "10".
3. Соедините на гистограмму тега Count_Bottle из таблицы тегов S7-300 Master>PLC Tags>**Markers**.
4. Откройте раздел Scales и измените числа делений гистограммы на "10".
22) **Рецепты**. Создание блока данных для передачи рецептов в программе контроллера. Создание рецептов и элементов рецепта в HMI-проекте. Отображение рецептов на экране процесса и передача рецепта в программу контроллера.
Создание рецептов приготовления сока
Для создания рецепта выполните следующие действия:
1. На дереве HMI-проекта откройте раздел **Recipes**. При этом открывается диалоговое окно создания рецептов. В этом окне выполните двойной щелчок на строке **Add new**. Появляется окно создания рецепта:

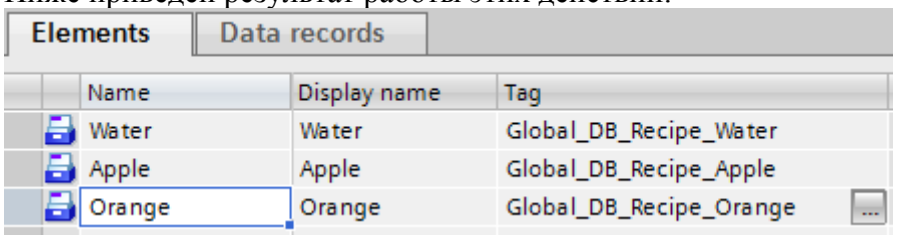


2. В столбце Name введите имя рецепта Recipe_Beverage, а в столбце Display name введите имя для отображения Beverages.

Создание элемента рецепта

1. Для создания элемента рецепта выберите вкладку **Elements** и выполните двойной щелчок на строке **Add new**. В столбцах Name и Display name введите имя элемента рецепта Water.
2. Соедините элемент рецепта с соответствующим тегом глобального блока данных. Для этого на поле Tag откройте блок данных Global_DB и выберите оттуда тег Recipe_Water.
3. Аналогично создайте второй элемент рецепта со следующими свойствами
– Имя: Apple
– Имя для отображения: Apple
– Тег: Recipe_Apple из глобального блока данных Global_DB.
4. Аналогично создайте третий элемент рецепта со следующими свойствами: Имя: Orange, Имя для отображения: Orange, Тег: Recipe_element_orange из глобального блока данных Global_DB.

Ниже приведен результат работы этих действий:

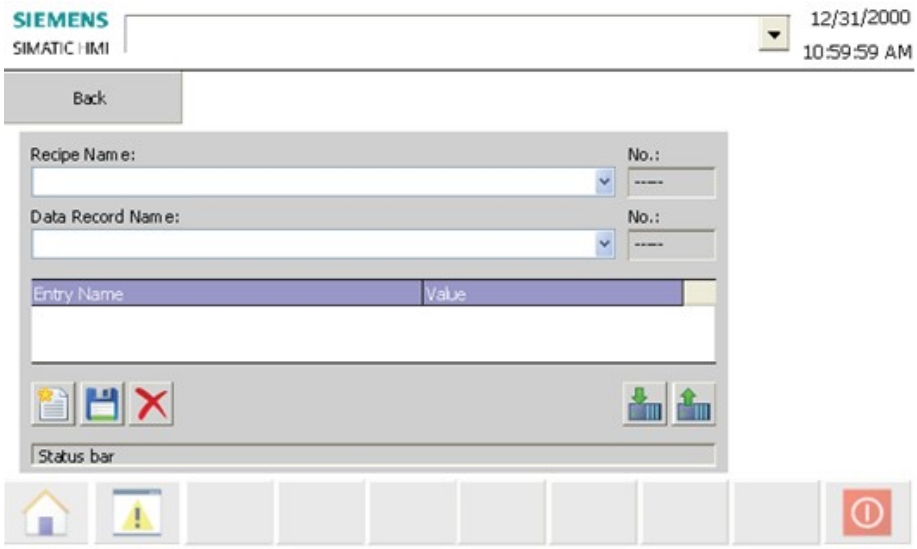


Создание данных для рецепта

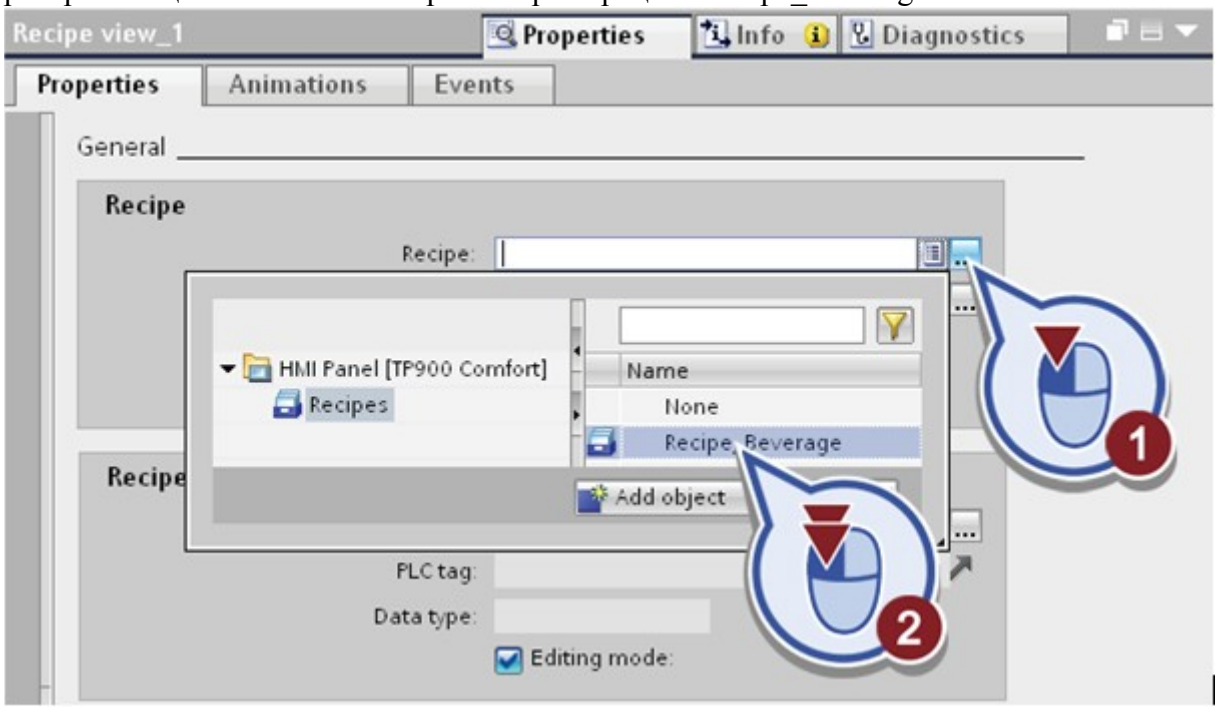
Для создания состава рецепта выполните следующих действий:
1. Откройте вкладку **Data records** и выполните двойной щелчок на строке <Add new>.
2. В полях Name и Display name введите Recipe_Water:
3. В поле Water введите число 10000, а другие ингредиенты рецепта Apple и Orange оставляйте значение "0".

Просмотр рецептов

1. С помощью дерева HMI-проекта откройте экран **Recipes** и из середины экрана удалите текстовое поле.
2. Далее открывайте панель инструментов и выберите в нем палитру объектов **Controls**. Выберите графический объект **Recipe view** и разместите его на экране.



3. Выберите объект **Recipe view** и откройте окна свойств. Во вкладке General из раскрывающегося списка Recipe выберите рецепт Recipe_Beverage:



Таким образом, в режиме исполнения вы можете переходить на экран Recipes и выбрать желаемый рецепт. Для обратного перехода на экран Production используйте кнопку Back.

23) **Тренды**. Отображение трендов с помощью элемента управления Trend view. Настройка Trend view и привязка на него тегов проекта. Масштабирование трендов по горизонтальным и вертикальным осям.

Назначение

Окно тренда позволяет четко отображать постоянно изменяющиеся значения процесса. В случае, когда процесс изменяется медленно, тренд может визуализировать события, которые уже произошли, для оценки трендов последовательности операций. С другой стороны, тренд может отображать значения процессов, которые изменяются быстро.

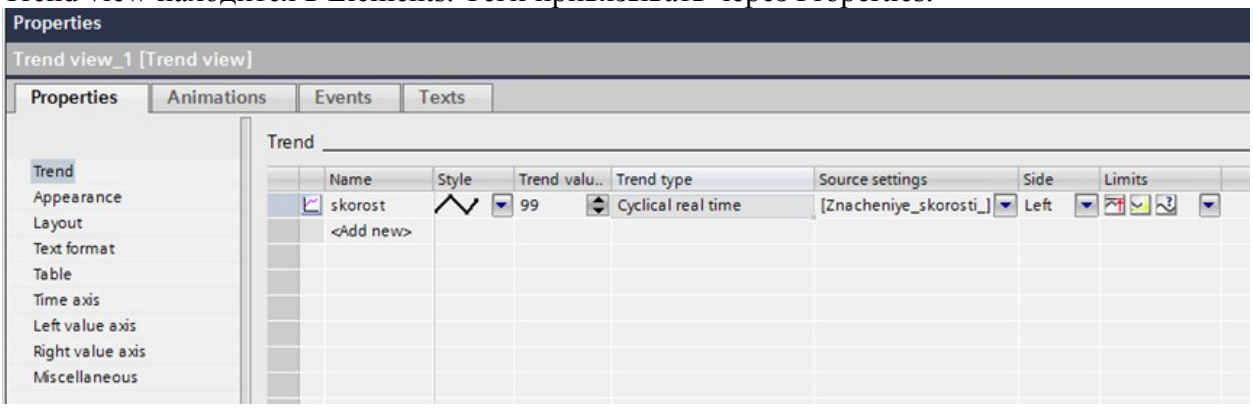
Определяемые свойства

В окне тренда одновременно может отображаться несколько трендов. Для них задаются следующие свойства:

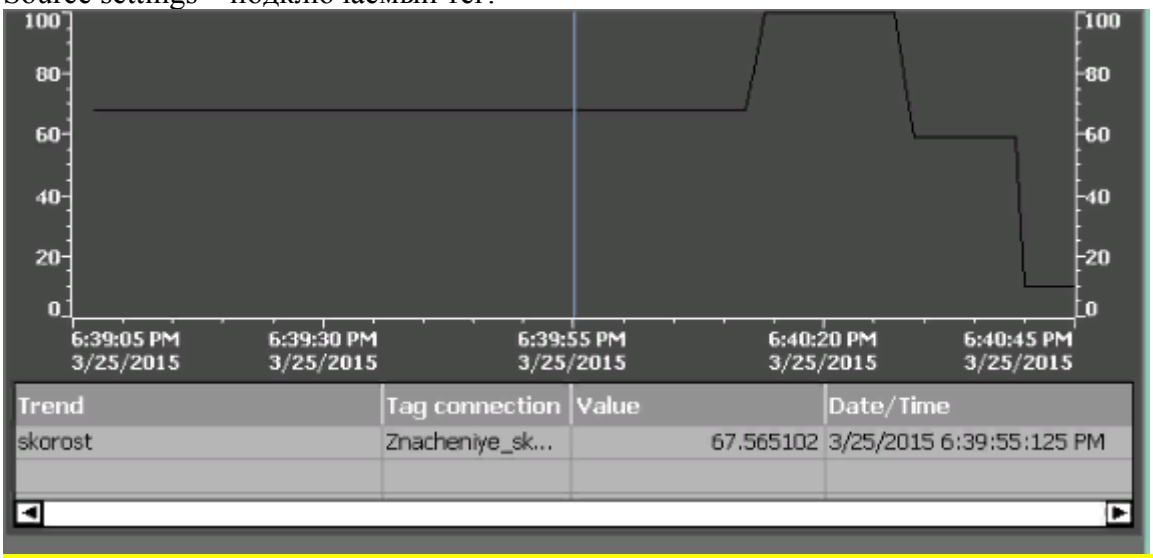
- Trend Type [Тип тренда]: Realtime trend [Тренд реального времени] или History trend [Исторический тренд]
- Trigger [Опрос]: Pulse [Пульсовый] или Bit [По биту]
- Граничные значения: Верхнее и нижнее граничные значения
  На MP 370 можно задать цвет значений, которые равны или превышают заданные граничные значения.
- Color [Цвет]: Uncertain status [Неопределенное состояние]

  Неопределенное состояние – это области окна тренда, для которых неизвестны значения из-за отсутствия связи с устройством. Эти области выделяться другим цветом.

Trend view находится в Elements. Теги привязывать через Properties.

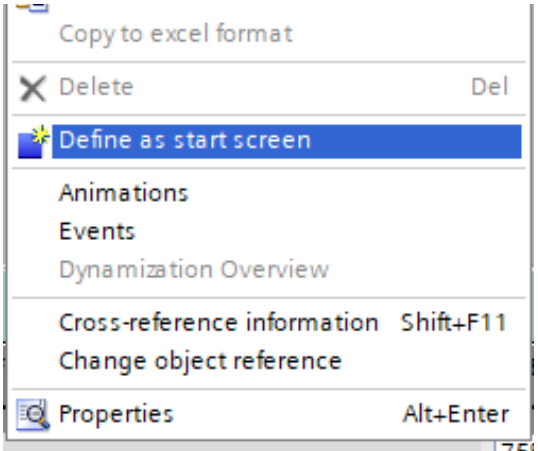


Source settings – подключаемый тег.



24) **Создание HMI-проектов со многими экранами процесса**. Реализация переключения из одного экрана процесса на другой экран процесса.

Для создания нового экрана выполните следующие действия :

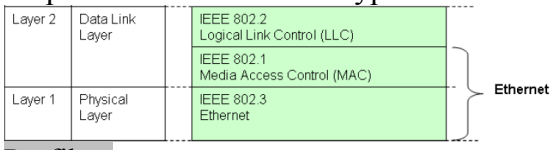


1. Щелкните правой кнопкой мыши на графическом редакторе, отображаемом в окне проекта, в котором будет добавлен новый объект.
2. Выберите в контекстном меню команду "Add screen [Добавить экран]".
3. Новый экран появляется в окне проекта и отображается в рабочей области.

Можно определить как стартовый экран.

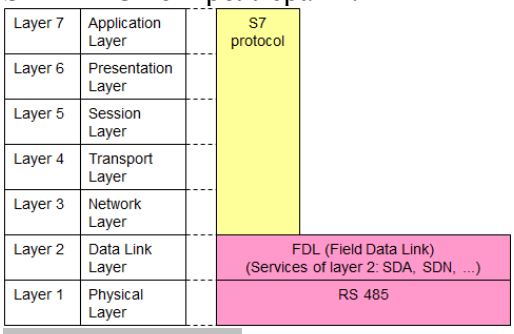
Перхабс: http://razrab.ru/viewtopic.php?f=17&t=754#p8802

25) **Интерфейсы промышленных сетей**:
MPI(многоточечный интерфейс) – подсеть малой протяжённости и с малым количеством абонентов для полевого и цехового уровня. Интерфейс способен объединять несколько точек в SIMATIC S7. Разрабатывался как интерфейс для устройства программирования(PG) и задумывался для соединения нескольких CPU между собой или с PG для обмена небольшими объёмами данных. Всегда сохраняет последнюю параметризацию относительно скорости передачи, номера абонента и наивысшего адреса MPI, в том числе после полного стирания памяти, исчезновения напряжения и стирания параметризации CPU.
Ethernet является стандартом для локальных сетей (LAN) в пределах офиса организации. Общими характеристиками этой сети являются:
• Международный стандарт: IEEE 802.3
• Простой и стандартизованный способ проведения сети
• Является основой для протоколов более высокого уровня (TCP/IP, UDP, …)
• Простое соединение для беспроводных сетей.
Это позволяет реализовать следующую функциональность:
• Несколько пар станций соединены с друг с другом в одно и то же время. Каждое соединение имеет полную пропускную способность.
• Трафик передачи локальных данных остается локальным. Только данные других подсетей передаются с помощью переключателей/роутеров.
Модель ISO/OSI
Ethernet включает в себя уровень 1 и уровень 2 модели ISO/OSI:
• Уровень 2: управление доступом и адресация (MAC адреса . Media Access Control – уникальный идентификатор, который присваивается любому активному сетевому оборудованию еще на заводе. MAC адрес есть у сетевых карт компьютеров и ноутбуков, у роутеров, у Wi-Fi модулей смартфонов и планшетов.
MAC адрес состоит из шести байт и позволяет точно определить оборудование, которое должно получить каждый конкретный пакет данных и отправить этот пакет именно туда, куда нужно).
• Уровень 1: Физический уровень.

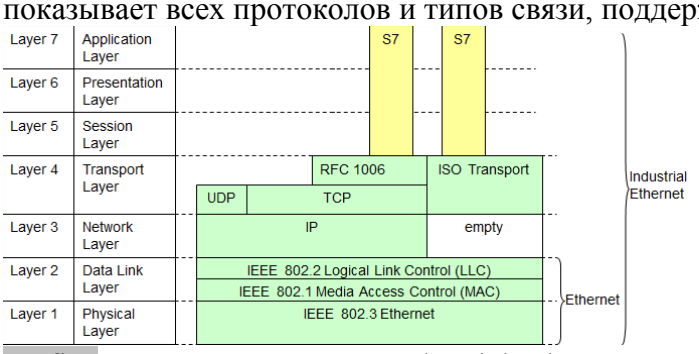


Profibus это сеть для полевого и цехового уровня в открытой, независимой от изготовителя системе связи SIMATIC.
PROFIBUS предлагается в двух вариантах:
1. в качестве полевой шины PROFIBUS-DP для быстрого циклического обмена данными и PROFIBUS-PA для организации связи в областях, требующих обеспечения взрывобезопасности

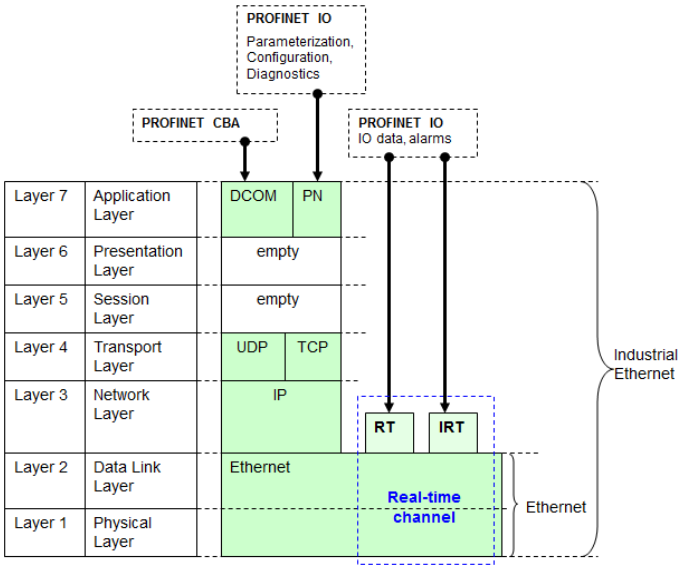
2. в качестве PROFIBUS (FDL или PROFIBUS-FMS) для быстрой передачи данных между равноправными партнерами по связи на цеховом уровне.
PROFIBUS является международным стандартом сети по шинной архитектуре.
Механизм связи между полевыми устройствами и контроллерами: по принципу Master/slave.
На следующем рисунке показаны все протоколы и типы соединений поддерживаемых SIMATIC контроллерами.



Industrial Ethernet является вариантом Ethernet подходящим для промышленных приложений. Кроме характеристик Ethernet, в случае Industrial Ethernet имеются следующие дополнительные характеристики:
• соединение в различных помещениях: офисы и производство
• использует возможностей IT стандартов (известных из офисного сектора – браузеры, e-mail, …) в автоматизации
• оптимизированное соединение между компонентами автоматизации и одновременная связь по TCP / IP (открытый стандарт).
Industrial Ethernet охватывает уровней от 1 до 7 модели ISO/OSI. На следующем рисунке показывает всех протоколов и типов связи, поддерживаемых SIMATIC–контроллерами.



Profinet является открытым Industrial Ethernet стандартом для автоматизации. Profinet основан на Industrial Ethernet. Profinet имеет два варианта реализации:
• PROFINET IO: концепция автоматизации для реализации модульных приложений путем интеграции распределенных модулей ввода/вывода с помощью связи реального времени.
• PROFINET CBA: компонентный модель для задач автоматизации, который основан на распределенных компонентах и подфункциях.
Profinet связь основан на Ethernet связь. Они различаются между тремя коммуникационными каналами или соответственно тремя уровнями производительности.



Поддержка контроллеров SIMATIC S7 этих интерфейсов и соответствующие обозначения контроллеров.