**Архитектура программы для ПР102 (управление ВДП)**

**Сверка источников**

Проектная архитектура разработана на основе изучения руководства по эксплуатации системы **VDP v1.55** и технической документации контроллера **ОВЕН ПР102-24.2416.16.2**. В ней учтены все режимы работы и требования, описанные в указанных источниках:

* **Режимы работы:** Основной автоматический режим – пастеризация – включен всегда. Дополнительно поддерживаются опции ручного управления нагревом/охлаждением, режима CIP-мойки, режимов розлива (непрерывного и порционного) и поэтапного (рецептурного) режима. Эти режимы соответствуют перечисленным в руководстве VDP v1.55:
  + Пастеризация с последовательными этапами нагрев – выдержка – охлаждение – выравнивание – хранение;
  + Ручной режим для прямого управления нагревателями, охлаждением и мешалкой;
  + Мойка (CIP) ёмкости, включая полную мойку и укороченные программы (щелочная, кислотная, ополаскивание);
  + Режим розлива: непрерывный розлив продукта и дозированный розлив порциями;
  + Поэтапный (рецептурный) режим приготовления продукта (например, сыра) с ручным переходом между шагами процесса.
* **Аппаратная конфигурация ПР102:** Контроллер имеет 16 дискретных входов (24 В DC), 8 универсальных аналоговых входов (AI, поддержка датчиков 0–10 В, 4–20 мА, термисторы), 14 транзисторных выходов NPN и 2 аналоговых выхода (0–10 В / 4–20 мА). Такая компоновка соответствует модификации **ПР102-24.2416.16.2**. В проекте используются два аналоговых входа AI1 и AI2 для термисторных датчиков NTC (контроль температуры продукта и рубашки), что согласуется с документацией ПР102 (поддержка NTC 10k) и требованиями задания. Управление нагревом реализуется через транзисторные DO (для контакторов/соленоидов нагрева) и через аналоговый выход AO (для ПИД-регулирования тиристорного модуля) – данная возможность предусмотрена опцией использования ПИД и твердотельных реле. Частотный преобразователь мешалки подключается по Modbus RTU (RS-485-2) в режиме Master, а связь с HMI/SCADA осуществляется по Modbus (RS-485-1 или Ethernet) в режиме Slave, согласно техническому описанию ПР102 (наличие двух интерфейсов RS-485 в модификации .2) и заданию.
* **Карта Modbus-регистров SCADA:** Архитектура обеспечивает полное соответствие переменных внутренней программы карте регистров Modbus, используемой верхним уровнем (SCADA/HMI). Согласно карте переменных панели, ключевые параметры привязаны к следующим Modbus-регистрам контроллера:
  + **Регистр 500** – маска аварий (16 бит), включает биты обрыва/КЗ датчиков температуры продукта и рубашки, аварии мешалки, ТЭНа, пропадания питания в авторежиме, ошибки ПИД-настройки и др. (всего 16 сигналов, см. ниже).
  + **Регистр 501** – температура продукта (Int16, шкала 0,1 °С; например, 25,5 °C → значение 255).
  + **Регистр 502** – температура рубашки (Int16, 0,1 °С, поддерживает отрицательные значения; например, -1,5 °C → 65521).
  + **Регистр 503** – этапы мойки CIP (битовая маска) с расшифровкой этапов для полной, щелочной, кислотной моек и ополаскивания.
  + **Регистр 504** – этапы пастеризации (битовая маска); биты 0–4 соответствуют этапам 1–5 (нагрев, выдержка, охлаждение, выравнивание, хранение), бит 5 – признак завершения пастеризации.
  + **Регистр 505** – состояния выходов (битовая маска 14 задействованных DO). Здесь каждому биту сопоставлен физический выход: циркуляционный насос/паровой клапан (бит0), клапан охлаждения (1), мешалка вперёд/реверс (2/3), насос розлива (4), нагреватели (5–7 для трёх групп тэнов), клапаны ХВС/ГВС (8/9), клапаны подачи щёлочи/кислоты (10/11), насос мойки (12) и клапан слива (13). Биты 14–15 зарезервированы.
  + **Регистр 506** – заданная частота мешалки (Word 16-bit, в единицах 0,1%, отображается как % от 50 Гц). Например, при уставке 25,0 Гц значение регистра = 500 (что соответствует 50,0% от 50 Гц). Диапазон 0–1000 соответствует 0–100,0% (50 Гц), при необходимости предусмотрен запас до 5000 (возможность задать до 250 Гц при 500,0%).
  + **Регистр 507** – режим отображения времени (код 0/1/2) для индикации оставшегося времени мойки, этапа либо прошедшего времени.
  + **Регистр 508** – время мойки, минуты (Word, отображается в минутах в соответствии с режимом 507).
  + **Регистр 509** – время мойки, секунды (Word, секундах по режиму 507).
  + **Регистр 510** – уставка температуры нагрева продукта (Word, 0,1 °С; например, 45,5 °C → значение 455).
  + **Регистр 511** – заданное время нагрева (минуты, Word).
  + **Регистр 512** – температура начала плавного нагрева продукта (Word, 0,1 °С; например, 35,5 °C → 355).
  + **Регистр 513** – дифференциал температуры рубашки при плавном нагреве (Word, 0,1 °С; напр. 1,5 °C → 15).
  + **Регистр 514** – время выдержки продукта (Word, секунды).
  + **Регистр 515** – дифференциал нагрева при выдержке (Word, 0,1 °С; напр. 1,5 °C → 15).
  + **Регистр 516** – температура охлаждения продукта (Word, 0,1 °С) – **TBD** (приведён далее в документе).
  + **Регистр 517** – относительная мощность нагревателей (Word, 0,01%; напр. 10,12% → значение 1012).
  + **Регистр 518** – статус системы (битовая маска), бит0 которого сигнализирует ручной (поэтапный) режим; остальные биты зарезервированы под прочие состояния.

Все указанные регистры и их значения были перепроверены с текстом задания и картой переменных панели. Таким образом, внутренняя программа экспонирует необходимые данные и принимает команды в точном соответствии со SCADA-интерфейсом. Масштабирование переменных выбрано согласно требованию: температуры хранятся в десятых долях градуса (0,1 °С) как знаковые int16, процентные величины нормированы в долях процента (0,1% или при необходимости 0,01%), частота мешалки – в 0,1% от 50 Гц. Такой подход обеспечивает совместимость с HMI/SCADA без дополнительных преобразований.

* **Наменование переменных:** В проекте принят единый стиль имен, соответствующий рекомендациям задания. Используются префиксы, отражающие тип и назначение переменной: DI\_/DO\_/AI\_/AO\_ для физических дискретных/аналоговых входов-выходов, PV\_ для измеренных технологических параметров (process value), SP\_ для уставок (setpoint), CMD\_ для команд операторов, MODE\_ и STEP\_ для режимов и этапов, STAT\_ для статусных флагов, ALM\_ для аварий, MB\_ для переменных, отображаемых в Modbus-регистрах, и т. п. Глобальными сделаны лишь переменные, участвующие во внешнем обмене или связывающие модули между собой, тогда как внутренняя логика каждого функционального блока использует локальные переменные. Параметры, подлежащие сохранению при перезагрузке (Retain) – например, уставки, настройки PID, счетчики доз – помечены как энергонезависимые и имеют предустановленные значения по умолчанию (паспортные или рекомендованные). Все эти правила отражены в таблицах переменных и CSV-файлах, подготовленных для импорта.

Таким образом, проверка исходных документов подтверждает, что разработанная архитектура полностью покрывает требования по режимам управления ванной длительной пастеризации, аппаратным ресурсам контроллера ПР102 и интерфейсу SCADA/HMI. В следующих разделах детально описана структура программы, логика переходов между состояниями, состав переменных и другие аспекты реализации.

**Архитектура: концепция, структура модулей, интерфейсы**

Архитектура программы организована модульно, с чётким разделением ответственности каждого блока. Ниже перечислены основные подсистемы (менеджеры) и их функции, а также взаимодействие между ними через глобальные переменные и интерфейсы:

**Менеджер режимов (Mode Manager)**

Менеджер режимов отвечает за выбор и координацию глобальных режимов работы системы. В любой момент времени активен только один из основных режимов:

* **Автоматическая пастеризация** (основной цикл с этапами).
* **Ручной режим** (прямое управление агрегатами).
* **Мойка (CIP)** – может запускаться как полная или сокращённая программа.
* **Розлив продукта** – в режиме непрерывного или порционного дозирования.
* **Рецептурный (поэтапный) режим** – последовательное выполнение шагов по заданному рецепту.

Менеджер режимов следит за командами оператора на запуск/остановку режимов и за исключающими условиями. Он получает сигналы от HMI/SCADA:

* CMD\_StartPasteur, CMD\_StopPasteur – команды на запуск/останов пастеризации;
* CMD\_StartCIP\_\* (для каждого вида мойки), CMD\_StopCIP – запуск/останова CIP;
* CMD\_StartDispCont/StopDispCont – запуск/стоп непрерывного розлива;
* CMD\_StartDispDose/StopDispDose, CMD\_NextDose – управление дозированным розливом;
* CMD\_StartRecipe, CMD\_StopRecipe – запуск/отмена рецептурного режима (если предусмотрено).

Получив команду на запуск режима, менеджер проверяет приоритеты и условия:

* Не допустить одновременную активность несовместимых режимов. Например, если идёт пастеризация, запрос на CIP игнорируется или ставится в очередь, и наоборот – сначала нужно выйти из текущего режима.
* При необходимости, останавливает предыдущий режим (корректно завершает процессы) перед переключением. Например, если оператор принудительно переключился с пастеризации на ручной режим, менеджер отменяет цикл пастеризации (с фиксацией "преждевременного останова") и только затем включает ручное управление.

Менеджер устанавливает глобальную переменную состояния режима, например MODE\_Current (кодировка режима) и/или отдельные флаги (MODE\_PasteurActive, MODE\_ManualActive, и т.д.). Остальные модули считывают эти флаги для понимания текущего контекста. Также Mode Manager передаёт в подчинённые модули параметры запуска: например, при старте CIP он сообщает тип мойки (полная, щёлочь, кислота, ополаскивание) в переменной CMD\_CIP\_Type, при старте дозированного розлива – начальную порцию и т.п.

**Интерфейсы:** Менеджер режимов взаимодействует с:

* **HMI/SCADA:** через переменные команд CMD\_\* (глобальные, R/W). Например, HMI-панель устанавливает CMD\_StartPasteur = 1 при нажатии кнопки "Пуск пастеризации", менеджер режимов это отслеживает и запускает автоматический цикл, затем сбрасывает команду.
* **Другими менеджерами:** при активировании режима менеджер может инициировать/останавливать подчинённые процессы. Например, при старте пастеризации – запускает температурный менеджер на цикл пастеризации, при останове – посылает ему сигнал завершить цикл досрочно. Взаимодействие реализовано через глобальные статусы STAT\_\* и вызовы/функции (например, установка флага аварийного прерывания цикла).
* **Модулем аварий:** если режим завершён нештатно (авария, прерывание питания), менеджер режимов обновляет маску аварий (биты "преждевременный останов" и т.п. в регистре 500).

Визуально менеджер режимов можно представить как "диспетчер", который в зависимости от входных команд выбирает, какой из нижележащих модулей активен, и уведомляет остальные о необходимости блокироваться. Например, при входе в ручной режим – выдаётся сигнал остальным модулям на паузу автоматических процессов.

**Температурный менеджер (Temperature Manager)**

Температурный менеджер отвечает за контроль температур продукта и рубашки и за достижение температурных уставок согласно текущему этапу процесса. Он функционирует в основном в автоматическом режиме пастеризации, а также используется частично в режимах CIP (для контроля нагрева/охлаждения во время мойки, если требуется) и в режиме хранения продукта.

Основные функции:

* **Сбор данных с датчиков:** Менеджер читает значения с аналоговых входов AI1 (температура продукта) и AI2 (температура рубашки). Результаты хранятся в глобальных переменных PV\_TempProduct и PV\_TempJacket (в десятках °C). Эти переменные обновляются циклически и доступны другим модулям и SCADA. Предусмотрена обработка неисправностей датчиков: встроенная диагностика ПР102 определяет обрыв/КЗ термодатчиков, при обнаружении менеджер температур устанавливает соответствующие аварийные биты (0–3) в ALM\_Mask и может останавливать нагрев для безопасности.
* **Управление нагревом и охлаждением:** Температурный менеджер определяет требуемый режим теплообмена на каждом этапе:
  + **Этап нагрева продукта:** вычисляет отклонение ∆T = SP\_HeatTemp - PV\_TempProduct. Если продукт еще не достиг заданной температуры пастеризации, инициируется нагрев. Менеджер активирует контур нагрева (через регулятор нагрева, см. ниже) и следит за температурой рубашки. При наличии настроек плавного нагрева, после превышения порога SP\_SmoothTemp менеджер ограничивает перегрев рубашки: уменьшает максимальную температуру рубашки до температуры продукта (либо до SP\_HeatTemp + SP\_SmoothDiff), чтобы выровнять градиент и предотвратить перегрев. То есть, на начальном участке нагрева рубашка может быть доведена до максимума для быстрого разгона, затем, когда продукт достиг, например, 50 °С, разница температур рубашка–продукт поддерживается не более заданного дифференциала (например, 2 °С).
  + **Этап выдержки:** после выхода на температуру пастеризации менеджер удерживает продукт на этой температуре в течение заданного времени SP\_HoldTime. Он поддерживает температуру, подавая тепло по необходимости: если продукт начинает остывать, допускается догрев. При этом применяется дифференциал нагрева на выдержке SP\_HoldDiff – максимальное превышение температуры рубашки над продуктом, чтобы не допустить перегрева во время выдержки. Например, если SP\_HoldDiff = 1,0 °C, рубашка будет включаться лишь когда продукт остынет примерно на 1 °C ниже уставки, и греть рубашку не более чем на 1 °С выше температуры продукта.
  + **Этап охлаждения:** по истечении времени выдержки менеджер переключается на охлаждение продукта до заданной температуры SP\_CoolTemp (температуры готового продукта). Он подаёт команду на открытие охлаждающего контура (клапан холодной воды DO\_ValveCold) и контролирует скорость охлаждения. Обычно мешалка продолжает работать для равномерности. Когда температура продукта снизилась до SP\_CoolTemp, менеджер закрывает клапан охлаждения.
  + **Этап выравнивания температур:** после прекращения активного охлаждения вводится пауза выравнивания. Система ждёт, пока температура рубашки и продукта максимально сблизятся (рубашка могла остаться чуть холоднее или теплее). Фактически, выравнивание – короткий этап, в течение которого нагрев и охлаждение выключены, а продукт может слегка прийти в равновесие с окружающей средой или рубашкой. Временной критерий – либо фиксированная задержка (например, несколько минут), либо достижение минимального градиента между рубашкой и продуктом (практически – рубашка приблизилась к продукту с точностью до, скажем, 0.5 °С). В руководстве указано, что выравнивание начинается после закрытия клапана охлаждения; в нашей реализации мы используем таймер выравнивания или контроль ∆T.
  + **Этап хранения продукта:** если в настройках включен режим хранения (например, PAR\_StoreEnabled), по завершении выравнивания температур менеджер не отключает систему, а переходит в режим долгосрочного хранения. Здесь цель – удерживать продукт на конечной температуре SP\_CoolTemp в течение неопределённого времени (например, для созревания продукта или поддержания охлаждения). Менеджер может использовать **режим охлаждения** для удержания (если продукт склонен нагреваться от окружающей среды) либо **нагрев** (если продукт, напротив, может переохладиться – менее вероятно). В стандартной конфигурации хранение после пастеризации означает поддержание охлаждением: включение охлаждения при превышении SP\_CoolTemp на некий гистерезис и отключение при снижении до нормы (подобно холодильнику). Эта опция (также называемая "режим поддержания температуры охлаждением") отражена в параметрах. Менеджер температур в режиме хранения периодически проверяет температуру продукта и включает/выключает охлаждение по необходимости. Если хранение не активно, процесс пастеризации считается завершённым сразу после этапа выравнивания (менеджер режимов получит сигнал об окончании цикла).

Во всех вышеперечисленных этапах температурный менеджер выдаёт **задания** регулятору нагрева и исполнительным механизмам:

* Команда **Heat\_Demand** (требование нагрева) – булевый или аналоговый сигнал, отражающий необходимость подачи тепла. Например, во время нагрева Heat\_Demand = 100% (максимум) до достижения SP; на этапе выдержки Heat\_Demand = 0 или малый процент, если температура превышает уставку, и = 100% когда падает ниже уставки на дифференциал.
* Команда **Cool\_Demand** – требование охлаждения (включения холодной воды). Во время активного охлаждения Cool\_Demand = 1, иначе 0.

Температурный менеджер также управляет **мешалкой** в автоматических циклах с целью равномерности температуры:

* На нагреве и выдержке мешалка, как правило, должна работать постоянно (либо циклически). В параметрах пастеризации оператор может выбрать режим работы мешалки (непрерывно или циклично) и задать период цикла. Менеджер температур (или отдельный подблок) реализует эту логику: если выбран циклический режим, мешалка включается и выключается с заданным интервалом (например, 30 с работа, 30 с пауза). Если непрерывный – держит всегда включенной. В этап охлаждения мешалка тоже рекомендуется (для ускорения охлаждения), а на хранении может быть отключена или периодически включается для перемешивания.
* В CIP-режиме тоже используется мешалка: при циркуляции моющих растворов мешалка обычно включена, чтобы поток обмывал стенки. В полном цикле CIP мешалка включается на этапах щёлочной и кислотной мойки. Наш температурный менеджер (либо сам CIP-менеджер) будет включать мешалку на этих этапах. Скорость мешалки во время мойки устанавливается максимальной заданной (в сервисных настройках может быть ограничение скорости).

**Интерфейсы:** Температурный менеджер получает:

* От **менеджера режимов** – информацию о текущем этапе/режиме и соответствующие уставки: например, в режиме пастеризации менеджер режимов указывает, что сейчас активен этап №1 (нагрев) и предоставляет значения SP\_HeatTemp, SP\_SmoothTemp, SP\_SmoothDiff и пр. (на практике уставки хранятся глобально, менеджер режимов только запускает/сбрасывает таймеры). При переходе на следующий этап (выдержка, охлаждение и т.д.) менеджер режимов сигнализирует температурному менеджеру изменить целевое SP и тип регулирования.
* От **датчиков (аппаратно)** – через системные АЦП контроллера. В программе это представлено либо системными переменными ADC, либо результатами преобразования модулей OwenLogic. Температурный менеджер периодически (например, раз в цикл PLC) обновляет PV\_TempProduct и PV\_TempJacket. Предусмотрена фильтрация помех (скользящее среднее на несколько отсчётов) при необходимости, чтобы исключить ложные скачки, которые могли бы вызвать дребезг в управлении нагревом/охлаждением.
* **Регулятор нагрева и менеджер исполнительных механизмов:** Температурный менеджер не управляет выходами напрямую, а выставляет цели: он передает рассчитанные Heat\_Demand и Cool\_Demand соответствующим модулям. Например, при Heat\_Demand > 0 менеджер нагрева должен активировать нагреватели, при Cool\_Demand = 1 исполнительный модуль должен открыть клапан охлаждения.

**Регулятор нагрева (Heating Controller)**

Данный модуль отвечает за непосредственную реализацию алгоритма **регулирования нагрева** продукта. Он получает от температурного менеджера требуемую степень нагрева и решает, каким образом задействовать нагревательные элементы (ТЭНы или пар) и соответствующие выходы (цифровые или аналоговые). Поддерживаются два принципа работы:

* **Двухпозиционное регулирование (гистерезисное или ШИМ):** применяется при управлении нагревом через дискретные выходы (например, контакторы ТЭНов или соленоид парового клапана). В простейшем случае реализован гистерезис: регулятор включает нагрев (DO) при температуре ниже уставки и отключает при превышении уставки на некоторый гистерезис. Точность поддержания определяется шириной дифференциала. Для повышения стабильности возможно использование широтно-импульсной модуляции: регулятор при малом отклонении от уставки не отключает нагрев полностью, а модулирует сигнал (например, включен 2 с, выключен 8 с и т.п.), то есть реализует двухпозиционный ШИМ. Однако при работе с мощными ТЭНами через контакторы чрезмерно частое включение нежелательно, поэтому скорее будет использован классический гистерезис. Параметры дифференциала заданы настройками (SP\_HoldDiff, SP\_SmoothDiff и т.д.).
* **Непрерывное ПИД-регулирование (PID):** используется при наличии аналогового канала управления мощностью – в нашем случае, аналогового выхода AO на тиристорный модуль для группы ТЭНов. Если контроллер оснащён такой опцией (что подразумевается особой модификацией или специальной настройкой), регулятор нагрева функционирует как PID-контроллер температуры продукта. Он вычисляет отклонение (error = SP - PV) и формирует управляющее воздействие (0–100%), которое преобразуется в аналоговый сигнал 0–10 В (или 4–20 мА) на AO. В программе задаются коэффициенты П, И, Д (ретентивные переменные PAR\_PID\_Kp, Ki, Kd), которые можно настроить эмпирически или через автоподстройку (в меню предусмотрен автонастройка ПИД). PID-регулятор обеспечивает плавное поддержание температуры с минимальными колебаниями и превышениями.

В зависимости от конфигурации оборудования, регулятор нагрева выбирает режим работы:

* Если активна опция **“клапан пара вместо тэнов”** (PAR\_UseSteam = True), или не задан аналоговый выход, то используется **дискретный режим:** DO\_Heater (или DO\_SteamValve) включается/выключается по гистерезису. Регулятор задает флаг DO\_HeatEnable = 1 или 0, основанный на требовании Heat\_Demand от температурного менеджера. Обычно, если Heat\_Demand = 100% (значительное недогрев), регулятор включает нагрев до достижения верхней границы уставки; если же Heat\_Demand низкий (например, поддержание), может задействоваться ПВМ с определенным циклом.
* Если доступен **аналоговый канал управления** и включена опция PID (PAR\_UsePID = True), то регулятор использует PID. Он вычисляет текущее необходимое процентное открытие нагрева (например, 0–1000 в 0,1%) и записывает его в переменную AO\_HeaterPower. Эту переменную мы привязываем к аналоговому выходу (см. таблицу сетевых переменных для Modbus, где, хотя регистр для управления AO не выделен, физически AO будет установлен согласно этому значению). Цифровые выходы ТЭНов при этом могут использоваться для второй и третьей группы нагревателей (см. ниже).
* **Комбинированный режим:** Возможна ситуация, когда первая группа нагревателей подключена через тиристор (PID по AO), а дополнительные группы – через контакторы (вкл/выкл). В опциях это отражено пунктами 11, 12, 13. В таком случае регулятор нагрева распределяет воздействие: например, PID ведет первую группу (регулирует 0–100%), а вторую/третью включает ступенчато при больших отклонениях. Реализация: если потребное PID-выход > ~80%, можно подключить вторую группу (DO\_Heater2) постоянно, если > ~160% – и третью, или подобная логика каскадного подключения. При снижении нагрузки – наоборот, сначала отрабатывается PID на первой группе, и только если он на минимуме, отключаются дополнительные группы. Такая ступенчатая схема позволит расширить мощность нагрева при больших объемах продукта, сохраняя точность вблизи уставки.

Регулятор нагрева тесно взаимодействует с менеджером исполнительных механизмов, который непосредственно управляет физическими выходами:

* В простом случае регулятор сам может устанавливать глобальные флаги типа DO\_Heater1\_Cmd, DO\_Heater2\_Cmd, ... или AO\_HeaterCmd – требуемые состояния/значения для выходов. Эти команды затем читает модуль исполнительных механизмов и проводит к реальным DO/AO.
* Альтернативно, регулятор может напрямую обращаться к аппаратным переменным (например, в среде OwenLogic возможна привязка переменной к выходу). Но для четкого разделения, предпочтителен вариант через промежуточные глобальные переменные команд, чтобы исполнительный модуль мог накладывать общие блокировки (например, отключить все нагреватели при аварии уровня воды, независимо от команд регулятора).

**Интерфейсы:**

* Вход: Heat\_Demand (от температурного менеджера, может быть дискретным или аналоговым сигналом, показывающим степень необходимости нагрева).
* Настройки: PAR\_UsePID, PAR\_UseSteam, PAR\_HeaterGroup2Enabled, PAR\_HeaterGroup3Enabled, PID\_Kp/Ki/Kd и др. – глобальные параметры, задающие режим работы регулятора. Например, если PAR\_UseSteam=True, регулятор знает, что нужно работать одним каналом (паровой клапан) вместо трех отдельных ТЭНов.
* Выход: набор сигналов управления нагревательными элементами:
  + CMD\_HeaterAO (целевое значение аналогового выхода на тиристор, 0–1000 условно),
  + CMD\_Heater1\_On, CMD\_Heater2\_On, CMD\_Heater3\_On (булевы команды включения групп ТЭНов, для контакторов).
  + Если паровой клапан, то CMD\_SteamValve\_On вместо вышеперечисленных. В реализации это может быть переиспользовано CMD\_Heater1\_On для Steam, поскольку электрические ТЭНы тогда не используются – но лучше сделать отдельный флаг для ясности.
* Обратная связь: от исполнительного модуля могут поступать сигналы об актуальном состоянии (например, STAT\_Heater1\_Fault – если сработала защитная автоматка на ТЭНе, но это скорее обрабатывается через DI и модуль аварий).

Регулятор также может иметь свой механизм **защиты/диагностики**:

* Контроль целостности ПИД-настроек. Например, если PID\_Kp=0 или другие некорректные значения, регулятор фиксирует ошибку настройки (устанавливает бит 7 в маске аварий).
* Контроль перегрева рубашки: если температура рубашки пошла значительно выше заданной (например, термодатчик вышел из строя или клапан пара залип), регулятор через аварийный модуль может инициировать аварийное отключение нагрева.
* Ограничение мощности: на основе параметра SP\_HeaterPower (мощность нагревателей в процентах) регулятор может оценивать максимальную скорость нагрева. Этот параметр может использоваться для справки (например, рассчитать прогноз времени нагрева) или для лимитирования выхода PID (например, чтобы PID не давал 100% дольше определённого времени без эффекта – хотя это скорее задача менеджера режимов, см. *время нагрева*).

**Менеджер исполнительных механизмов (Actuator Manager)**

Этот модуль осуществляет непосредственное управление всеми выходными устройствами системы – "приводами" (насосами, клапанами, нагревателями, мешалкой и т.д.) – в соответствии с командами, поступающими от остальных модулей. Его задача – централизовать контроль исполнительных элементов, обеспечить приоритеты, взаимоблокировки и безопасное состояние оборудования.

Основные функции:

* **Приведение команд к физическим выходам:** Менеджер исполнительных механизмов собирает команды управления от менеджера режимов, температурного менеджера, регулятора нагрева, CIP-менеджера и пр., и задаёт соответствующие состояния дискретных выходов DO\_\* и аналоговых AO\_\*. Практически, он обновляет переменные, привязанные к выходам контроллера (например, в OwenLogic каждый DOx может быть ассоциирован с булевой переменной).
* **Разрешение конфликтов и взаимоблокировка:** Если разные модули требуют противоположных действий на одном механизме, менеджер разрешает это по определённым правилам:
  + Пример: регулятор нагрева может подавать команду на включение нагрева CMD\_Heater1\_On = 1, но одновременно температурный менеджер в режиме охлаждения требует отключить нагрев. Менеджер исполнительных механизмов распознает, что сейчас активен режим охлаждения (глобальный флаг), и принудительно блокирует сигнал нагрева (т.е. не включает DO\_Heater1, даже если регулятор продолжает просить, пока режим охлаждения не завершится). Такая логика предотвращает одновременное нагревание и охлаждение.
  + Другой пример – мешалка: в автоматическом режиме управление мешалкой идёт от температурного менеджера (вкл/выкл или циклично), а в ручном режиме – от непосредственной команды оператора. Если оператор включил ручной режим мешалки, а система при этом была в автопастеризации, менеджер режимов отключит пастеризацию, или, если это не произошло, исполнительный менеджер приоритетно подчинится ручной команде (вручную включить мешалку) и временно игнорирует автоматическое управление. Однако логика Mode Manager скорее не допустит такой параллельности – ручной режим блокирует автопастеризацию целиком.
* **Реализация задержек и порядков включения:** Некоторые механизмы требуют соблюдения порядка или тайминга:
  + **Мешалка (двигатель) и частотник:** Если предусмотрен реверс мешалки (опция 8) и два выхода DO\_MixerFwd/DO\_MixerRev, менеджер запрещает мгновенный переключение направления. Он реализует последовательность: при смене направления сначала останавливает мешалку (снимает оба сигнала), выжидает паузу (например, 2–3 секунды для полной остановки двигателя и размагничивания), затем включает противоположный сигнал. Это предотвращает аварийные токи в моторе/частотнике.
  + **Насосы и клапаны CIP:** При переключении этапов мойки, необходимо закрывать одни клапаны прежде чем открывать другие, и не допускать одновременного закрытия всех выходов, чтобы избежать гидроударов. В графиках CIP указано, что окончание этапа происходит после отключения всех связанных элементов. Менеджер исполнительных механизмов обеспечивает: по истечении времени этапа CIP он проверяет, что клапаны ХВС/ГВС, насос и слив выключены, только затем разрешает начать следующий этап (открыть новые клапаны). Если какой-то элемент задерживается (например, клапан слив еще открыт), менеджер может выдерживать небольшую задержку.
  + **Нагреватели и охлаждение:** При переходе от нагрева к охлаждению (этап 2→3 пастеризации), желательно, чтобы нагреватели были отключены за некоторое время до открытия охлаждающей воды, чтобы исключить "борьбу" тепла и холода. Менеджер может вводить небольшой интервал между отключением нагрева и началом охлаждения (например, 5–10 с, если технологически нужно). Однако в данном процессе охлаждение обычно начинается после выдержки, где нагрев к тому моменту давно отключен, поэтому специальная задержка не требуется.
* **Контроль начальных и аварийных состояний:** Менеджер исполнительных механизмов обеспечивает, что при старте контроллера все выходы находятся в безопасном состоянии (все DO = Off, AO = 0). Он берет на себя и обесточивание при аварии: модуль аварий (см. ниже) может передать сигнал ALM\_GlobalStop или выставить соответствующую маску, и менеджер исполнительных механизмов в таком случае немедленно отключит нагреватели, насосы, закрывет клапаны – перевод оборудования в состояние минимальной опасности.
  + Например, авария по уровню воды в рубашке: как только ALM\_NoWaterJacket = 1 (бит 12), исполнительный менеджер незамедлительно выключит DO\_Heater\* и предотвратит их повторное включение до сброса аварии.
  + Аналогично, при аварии мешалки (бит 4) он отключит сигналы на частотник/контактор мешалки.
* **Согласование с физической обратной связью:** Если некоторым выходам соответствуют входные датчики для контроля их исполнения (например, есть датчик срабатывания контактора или датчик потока воды), менеджер исполнительных механизмов может сличать команду и фактическое срабатывание. В нашем проекте подобное предусмотрено для мешалки и нагревателей: входы DI\_MixerFault (X1) и DI\_HeaterFault (X2) от сигнализации частотника и от аварийного термовыключателя тэнов. Если, например, мешалка командована Вкл, а с частотника сразу пришёл сигнал "Fault" либо не пришло подтверждение "работает", менеджер отмечает эту ситуацию как авария мешалки.

**Интерфейсы:**

* Входы:
  + Команды от регуляторов/менеджеров: CMD\_\* переменные для всех исполнительных устройств. Например, CMD\_PumpCIP\_On, CMD\_Mixer\_On, CMD\_MixerRev\_On, CMD\_ValveCold\_Open, CMD\_ValveHot\_Open, CMD\_ValveAlk\_Open, CMD\_ValveDrain\_Open, CMD\_DispPump\_On и т.д. (Полный перечень см. таблицу глобальных переменных). Эти команды устанавливаются соответствующими функциональными модулями:
    - Менеджер режимов: высокоуровневые команды типа CMD\_Mixer\_ManualOn (включить мешалку в ручном режиме), CMD\_Dispense\_On.
    - Температурный менеджер: CMD\_Mixer\_AutoOn (мешалка в авто), CMD\_Cooling\_On (охлаждение).
    - Регулятор нагрева: CMD\_Heater1\_On/AO\_HeaterPower и пр.
    - CIP-менеджер: CMD\_ValveHot\_On, CMD\_ValveAlk\_On и т.д. на каждом этапе мойки.
  + Состояние глобальных режимов: MODE\_\* флаги, позволяющие понять контекст. Например, если MODE\_ManualActive=1, менеджер приводит команды преимущественно от ручного режима, игнорируя автоматические.
  + Аварийные флаги (от модуля аварий): например, ALM\_StopAll (глобальная авария), ALM\_NoWaterJacket, ALM\_MixerFault и т.п., чтобы знать, что должно быть принудительно отключено.
  + Физические входы: DI\_\* от датчиков оборудования – уровень воды, сигналы аварий от внешних устройств. Менеджер опрашивает их либо напрямую, либо через модуль аварий (в зависимости от реализации). Например, DI\_LevelJacket – если 0 (нет воды), менеджер блокирует нагрев.
* Выходы:
  + Управляемые переменные, привязанные к физическим DO/AO контроллера: DO\_PumpCirc, DO\_ValveCold, DO\_ValveHot, DO\_MixerFwd, DO\_MixerRev, DO\_PumpDispense, DO\_Heater1, DO\_Heater2, DO\_Heater3, DO\_ValveAlk, DO\_ValveAcid, DO\_PumpCIP, DO\_ValveDrain, а также AO\_HeaterPower%. Менеджер выставляет им значения на основе входных команд, но с учётом описанных приоритетов и блокировок.
  + Кроме непосредственного управления, менеджер формирует для SCADA сводку состояний: переменная MB\_OutStateMask (Modbus-регистр 505) обновляется менеджером исполнительных механизмов каждую итерацию согласно текущим значениям DO. То есть, битовая маска выходов составляеться из внутренних флагов DO, что позволяет диспетчеризации видеть, что сейчас включено.
* Обратные связи:
  + Менеджер может устанавливать статусы STAT\_\* для других модулей. Например, STAT\_HeatingActive – признак, что хоть один нагреватель включен (может использоваться температурным менеджером для оценки, или модулем аварий для контроля времени работы).
  + Сигналы о принудительных блокировках: например, STAT\_CoolingOverride=1 если менеджер выключил нагрев принудительно из-за охлаждения – это может быть полезно для логирования.

**Связь с частотным приводом (Modbus Master, VFD Manager)**

Частотный преобразователь мешалки (привод мешалки) подключён ко второму порту RS-485 ПР102, работающему в режиме Modbus Master. За обмен с ПЧ отвечает отдельный модуль – **VFD Manager**, который реализует опрос/управление по Modbus RTU.

Основные задачи:

* **Передача уставки частоты вращения:** Менеджер принимает от программы требуемое задание частоты мешалки – глобальную переменную MB\_SetMixerFreq (значение в 0,1% от 50 Гц, диапазон 0–1000). Эта переменная может устанавливаться оператором через HMI/SCADA (регистр 506) или автоматически быть равной 1000 (100%) по умолчанию. VFD Manager конвертирует это значение в формат, ожидаемый ПЧ, и записывает его в соответствующий регистр ПЧ (например, Holding Register для задания частоты). Точный адрес и формат настраивается по паспорту ПЧ; для конкретики: если ПЧ ОВЕН, то часто используется параметр с десятичной долей. В любом случае, MB\_SetMixerFreq = 500 соответствует 25 Гц, VFD Manager отправит 25.00 Гц на частотник.
* **Управление запуском/остановом и направлением:** Несмотря на наличие управления скоростью, для безопасного пуска двигателя обычно используются дискретные сигналы. В нашем оборудовании мешалка может управляться:
  + Через **DO3/DO4** (Forward/Reverse): эта схема применима, если ПЧ сконфигурирован на управление по дискретным входам для пуска. Тогда DO\_MixerFwd и DO\_MixerRev подключены ко входам ПЧ "Пуск вперед" и "Пуск назад". VFD Manager в таком случае может вообще не задействоваться для старта/стопа – это сделает исполнительный менеджер через DO напрямую (так безопаснее: при обрыве связи RS-485 привод всё равно остановится, так как команда приходит по проводу).
  + Либо полностью по Modbus: отправляя в регистр управления ПЧ командное слово (например, бит "RUN/STOP" и "Direction"). Это более гибко, но требует надежной связи. Если выбрана эта стратегия (опционально, PAR\_VFD\_ControlViaBus=True), VFD Manager будет при каждом цикле отправлять команду "Пуск" когда нужно включить мешалку.
  + В нашем проекте, учитывая наличие выходов DO3/DO4 и упоминание их в битовой маске, мы предполагаем **гибридный вариант**: **старт/стоп/реверс – через DO**, а **скорость – по Modbus**. Таким образом, VFD Manager только устанавливает частоту, а исполнительный модуль включает/отключает DO\_MixerFwd/Rev в соответствии с командами. Этот вариант повышает надёжность остановки (если PLC выключит DO, привод гарантированно остановится).
* **Опрос состояния привода:** VFD Manager периодически читает диагностические регистры ПЧ – состояние (работает/остановлен, авария), текущую фактическую частоту, ток, ошибочные коды. Не все эти данные требуются SCADA, но **авария ПЧ** крайне важна. Если запрос статуса выявил, что ПЧ в состоянии Fault, либо связь с ПЧ потеряна (нет ответа), VFD Manager информирует систему:
  + Устанавливает ALM\_MixerFault (бит 4 в авариях).
  + Опционально, может записывать код ошибки привода в какую-то переменную (для локального отображения на HMI).
  + Подает сигнал менеджеру исполнительных механизмов отключить мешалку (снять DO\_MixerFwd/Rev).
* **Настройки и калибровка:** Менеджер также может следить за ограничением максимальной скорости, если задан параметр PAR\_MaxMixerSpeed (например, 80% от 50 Гц). Если опция ограничения скорости (п.10) активна, VFD Manager при попытке выставить больше ограниченного значения, снижает MB\_SetMixerFreq до максимума. Также, если PAR\_UseVFD=False (опция 10 выключена), это означает, что, возможно, мешалка должна работать с фиксированной скоростью (например, если стоит двигатель без ПЧ). В таком случае VFD Manager может игнорировать переменные скорости, а менеджер исполнительных механизмов просто будет включать/выключать мотор на 100% через контактор K2 (предусмотрено схемой).

**Интерфейсы:**

* **Modbus RS-485 Master (порт 2):** VFD Manager использует встроенные функциональные блоки OwenLogic или шаблон обмена. В CSV Сетевые, Слот 2.csv (условно) были бы перечислены запросы к регистрам частотника (но по заданию эти детали опущены). Главное – привязка: переменная MB\_SetMixerFreq связана с записью в регистр частоты ПЧ, переменная статуса STAT\_VFD\_Word связана с чтением регистра состояния ПЧ и т.д. Период опроса настраивается (например, 100 мс для записи уставки при изменении, 500 мс для опроса состояния).
* **От других модулей:**
  + От менеджера исполнительных механизмов VFD Manager может получать сигнал на отключение выдачи уставки (например, если мешалка выключена, можно не слать 0 постоянно). Но, как правило, он просто всегда пишет текущую MB\_SetMixerFreq.
  + От Mode Manager/VFD settings: Если, например, выбрана реверсная работа (Manual Reverse), а управление направлением идет через шину, VFD Manager получит флаг CMD\_MixerReverse=True и выставит соответствующий бит команды.
* **К остальным:** VFD Manager выставляет STAT\_MixerRunning (на основе данных ПЧ) – может использоваться в HMI для индикации. А также может устанавливать ALM\_MixerFault напрямую (что дублируется модулем аварий).

**Связь SCADA/HMI (Modbus Slave)**

Модуль связи Modbus Slave обеспечивает обмен данными с внешней панелью оператора (Weintek) и системой диспетчеризации. В среде OwenLogic это реализуется посредством **сетевых переменных** (Network), привязанных к модбас-регистрам. Наша задача – гарантировать согласованность этих переменных с внутренними глобальными переменными программы.

Карта регистров уже описана в разделе сверки источников. Здесь уточним организацию модуля:

* **Отображение внутренних переменных в регистры:** Каждая нужная для внешнего чтения/записи переменная объявлена как **сетевая** (Metatype = Network) с указанием Modbus-адреса. Например, глобальная переменная MB\_Alarms (mask аварий) привязана к регистру 500, MB\_TempProduct – к 501, и т.д. (см. таблицу сетевых переменных). Вся актуализация данных происходит автоматически: при изменении глобальной переменной ее значение доступно по Modbus.
* **Обработка команд записи:** Для переменных, доступных на запись (R/W), необходимо предусмотреть реакцию программы. В нашем случае операционные команды (запуск режимов и пр.) **не привязаны** напрямую к Modbus-регистрам – они поступают от HMI как изменения отдельных бит или флагов, но не через стандартизированные регистры 500+. Обычно HMI Weintek работает как Modbus Master, читающий/пишущий и эти же регистры, и/или напрямую внутренние переменные. Однако исходное задание указывает конкретные регистры, в основном для мониторинга. Из них на запись скорее всего предназначены:
  + Настройки пастеризации (регистры 510–517) – SCADA может удаленно их менять (тип R/W).
  + Заданная частота мешалки (506) – тоже R/W, чтобы диспетчеризация могла задать скорость или чтобы HMI передавала сигнал оператора (кнопки "увеличить/уменьшить скорость" или ввод значения).
  + Режим отображения времени (507) – вероятно, R/W (панель сама может ставить значение 0,1,2 в зависимости от выбора оператора).
  + Остальные, вроде масок аварий и этапов, – только **Read** (только чтение), т.к. они формируются программой.

Модуль связи должен отличать эти случаи. В OwenLogic, если внешний мастер пишет в сетевую переменную, она автоматически обновится. Например, если SCADA запишет 600 в регистр 506, переменная MB\_SetMixerFreq примет значение 600. Наша программа (VFD Manager) это обработает в следующем цикле. То же с уставками – при записи нового значения в SP\_HeatTemp (регистр 510) изменится глобальная переменная, и температурный менеджер будет далее использовать новую уставку.  
Таким образом, для R/W переменных (частота, параметры пастеризации) достаточно, чтобы они были **Retain** (чтобы сохраниться между циклами питания) и использовались по месту. Дополнительно можно реализовать событие "изменение уставки" – например, при получении новой уставки сбрасывать какие-то промежуточные переменные (если нужно пересчитать что-то).

* **Очередь обмена:** ПР102 в режиме Slave поддерживает одновременный доступ панели и других клиентов. Панель Weintek подключается по Ethernet Modbus TCP (с указанным портом 8000). Вероятно, панель работает как gateway, но для нас важно обеспечить, чтобы обмен успевал за запросами. 19 регистров – небольшой объем, чтение их всех (функция 0x03) займет миллисекунды.  
  Мы оставляем адреса регистров **TBD** в настройках (для сопоставления, но предполагаем, что адреса начиная с 500 соответствуют указанным). Фактически, Modbus-адрес 500 означает номер регистра 500 (0-based addressing в тексте учтен).  
  В среде OwenLogic настройка состоит в добавлении всех сетевых переменных и указании их адресов. После этого контроллер сам отвечает на чтение 0x03 и запись 0x06/0x10 на эти адреса.
* **Дополнительные переменные для HMI:** Учитывая функциональность панели, в проекте могут быть глобальные переменные, не попавшие в стандартную карту 500+, но нужные панели. Например, **состояние системы** (регистр 518) бит0 = ручной режим. Мы настроили MB\_SystemStatus (регистр 518) – панель, видимо, использует бит0, чтобы, например, отображать что включен поэтапный режим.  
  Также, HMI может напрямую читать не только через Modbus, но и через **системные переменные Owen** (если панель подключается как OPC или специфичный драйвер). В рамках нашего проекта, для единообразия, считаем, что панель тоже использует Modbus. Поэтому любые дополнительные данные, нужные на панели (например, индикаторы уровня продукта, воды) нужно отразить в регистрах.  
  В карте 500+ мы видим косвенно упоминание: индикатор уровня продукта (использование датчика продукта) – панель скрывает/показывает элемент в футере. Для этого она, вероятно, читает напрямую дискретный вход или его отображение. Мы можем выдать этот сигнал, например, в свободный бит статуса системы (например, задействовать какой-то резервный бит 518 под "наличие продукта" или добавить сетевую переменную типа Coil). Но за неимением этого в задании, мы оставим логику отображения уровня продукта внутри панели (через прямое чтение DI или другого рег.).

**Интерфейсы:**

* **SCADA/Panel <-> PLC:** через сетевые переменные, перечисленные выше.
* **Логика программы <-> сетевые переменные:** в основном прямая привязка. Исключения:
  + Маски этапов (503 и 504) формируются в программе. Менеджер режимов/соответствующие модули устанавливают биты, а модуль связи просто отдает их. Запись в них извне не допускается (Read-only).
  + Маска аварий (500) – формируется модулем аварий, извне только чтение.
  + Время мойки (508/509) – эти регистры **только чтение**, задаются CIP-менеджером. 507 (режим отображения) – может записываться SCADA, но, скорее, это сугубо для локального интерфейса панели; SCADA обычно сама знает, какое время ей важно. Возможно, 507/508/509 предназначены *только* панели, а SCADA может не использовать.
* **Протоколы:** Поддерживается Modbus RTU (через RS-485-1) и Modbus TCP (через Ethernet, если опция есть). В тексте задания панель подключается по Ethernet Modbus TCP/IP, при этом контроллер ПР102 выступает Slave. В конфигурации OwenLogic мы настроим один сетевой Slave-устройство (Slot 1) – который охватывает указанные регистры. При необходимости, второй интерфейс RS-485 может быть тоже Slave для дублирования (например, для подключения резервной SCADA по 2-му порту).
* **Безопасность обмена:** Если связь с SCADA/панелью прервется, контроллер продолжит автономную работу. Поскольку панель в основном мониторит и задает параметры, потеря связи не должна влиять на контур управления. Все команды, полученные по Modbus, дублируются состоянием на контроллере. Например, если оператор нажал *Пуск* на панели и связь пропала, контроллер все равно запустил режим и завершит его. Однако потерю связи можно логировать: некий таймер "нет связи с HMI" – но у нас есть бит 15 аварий "связь с модулем расширения ПРМ", который в данном проекте в отсутствие внешнего модуля не используется. Его можно перепрофилировать под "нет связи с панелью", если был бы способ контролировать подключение (чаще это реализуется со стороны SCADA). Мы оставляем бит 15 зарезервированным (TBD).

**Механизм аварий и масок (Alarm Manager)**

Модуль аварий отслеживает все нештатные ситуации и формирует битовую маску аварий (глобальная ALM\_Mask, выводимая в регистр 500). Он же инициирует аварийные действия, такие как экстренная остановка оборудования или блокировка запуска циклов до устранения причины.

Какие аварии предусмотрены (по битам маски рег.500):

* **Bit 0/1 – Датчик температуры продукта: обрыв / короткое замыкание.** Определяется электроникой ПР102: при выходе сигнала за диапазон (например, >300 кОм или <0 Ом) контроллер может установить специальный признак. Мы реализуем программно: Temperature Manager, считвая PV\_TempProduct, проверяет флаг ошибки или нереалистичное значение (например, (65535 или 32767, или наоборот 0 при разрыве, в зависимости от настройки). При обнаружении устанавливается ALM\_SensorProduct\_Break или ALM\_SensorProduct\_Short. Обе эти аварии – **нефункциональные** (критические): при них программа:
  + Прерывает текущий цикл пастеризации/мойки (если шёл), т.к. продолжать без контроля температуры опасно.
  + Блокирует включение нагревателей (во избежание перегрева) и выдает оператору сигнал.
  + Требует устранения (замены датчика) и сброса аварии (вручную).
* **Bit 2/3 – Датчик температуры рубашки: обрыв / КЗ.** Аналогично продукту. Используется тот же механизм обнаружения. При аварии рубашечного датчика можно, теоретически, продолжать пастеризацию (если продуктовый датчик цел), но ухудшится контроль нагрева (ПИД не сможет следить за рубашкой). В исходном проекте такие аварии, вероятно, также прерывают цикл (с точки зрения алгоритма задано, что обрыв или КЗ датчика рубашки – критическая ситуация, хотя продукт не перегреется из-за ПИД, но система теряет резервный контроль). Мы примем консервативно: авария любого датчика температуры останавливает автоматический режим.
* **Bit 4 – Авария мешалки.** Причины: перегрузка/ошибка частотника, клин двигателя, отсутствие вращения. У нас она может сработать либо от сигнала DI\_MixerFault (если частотник имеет релейный выход аварии, подключенный к X1), либо от внутреннего мониторинга VFD Manager (нет ответа по Modbus, код ошибки). При этой аварии:
  + Если мешалка была нужна по процессу (например, на этапах пастеризации), процесс может продолжаться, но качество перемешивания падает – это нежелательно, но не опасно сразу. Тем не менее, вероятно, лучше приостанавливать процесс пастеризации, так как равномерность температуры не гарантируется. Руководство явно не оговаривает, но оператору точно подаётся сигнал об аварии привода. Мы сделаем: при ALM\_MixerFault, если пастеризация/мойка активны, они **ставятся на паузу** или завершаются досрочно (можно считать, как преждевременный останов, бит 13/14). В то же время, нагрев можно отключить (чтобы продукт не пригорал без мешалки).
  + Вручном режиме авария мешалки просто блокирует её включение.
  + Сброс аварии произойдёт, когда оператор устранит проблему (перезапустит ПЧ и нажмет сброс).
* **Bit 5 – Авария ТЭНа.** Это общее обозначение аварии нагревательной системы. Возможные причины: срабатывание защиты ТЭНов (например, термостат на рубашке, фиксирующий перегрев воды), перегрузка по току (автомат выключился) или отсутствие питания на ТЭН. В схеме есть вход X2 "Авария ТЭН", который, вероятно, приходит от термозащиты рубашки. При его срабатывании (DI\_HeaterFault = 1):
  + Модуль аварий ставит ALM\_HeaterFault = 1.
  + Менеджер исполнительных механизмов немедленно снимает все сигналы нагрева (DO\_Heater\* = 0).
  + Автоматический режим, если шёл, прерывается (т.к. продолжать пастеризацию невозможно без нагрева).
  + Оператору сигнализируется авария. Требуется проверить систему отопления (возможно, перегрев рубашки из-за отсутствия воды – тогда одновременно был бы ALM\_NoWaterJacket; либо отказ насоса циркуляции).
* **Bit 6 – Авария по потере питания в авторежиме.** Данный бит предназначен для фиксации факта внезапного обесточивания контроллера во время выполнения цикла пастеризации или CIP. Когда питание восстановится, контроллер сам не узнает, был ли он в режиме, но мы можем сохранить признак:
  + Реализация: в Retain-памяти есть флаг, например STAT\_AutoActive, который устанавливается =1 при старте пастеризации/мойки и сбрасывается при их нормальном завершении. Если при запуске контроллера обнаружено STAT\_AutoActive = 1 (т.е. до выключения он был в середине цикла), то модуль аварий выставляет ALM\_PowerFail = 1.
  + Этот бит – информативный для оператора, он свидетельствует, что цикл был прерван по питанию и, возможно, продукт недопастеризован. После восстановления система не будет самопроизвольно продолжать – требуется вмешательство (вручную решить, что делать с партией продукта).
  + При желании, можно реализовать автодоводку: к примеру, запомнить этап и время на момент питания, а после включения предложить оператору продолжить с того места. Но в рамках данного задания мы ограничимся фиксацией аварии.
* **Bit 7 – Авария ПИД-регулятора (неверно заданы коэффициенты).** Выставляется регулятором нагрева, если обнаружит некорректные параметры (например, все коэффициенты по нулям, или бесконечность). Пока этот бит активен, регулятор может переключиться в запасной режим (гистерезис) либо просто отключиться. Оператору следует проверить настройки ПИД. Сброс – либо автоматический после ввода корректных значений, либо вручную.
* **Bits 8–11 – Резерв.** Возможны будущие расширения (например, авария питания мешалки, перегрев двигателя, но в нашем проекте ничего не назначено).
* **Bit 12 – Нет уровня в рубашке.** Срабатывает от датчика уровня воды в рубашке (опция 15, вход, допустим, DI\_LevelJacket). Если рубашка недостаточно заполнена водой (датчик сухой) при включении нагрева, это опасно – ТЭНы могут перегореть. Поэтому:
  + Если сигнал "нет воды" появляется (или был при старте), модуль аварий мгновенно ставит ALM\_NoWater = 1.
  + Менеджер исполнительных механизмов останавливает нагрев, циркуляционный насос и выводит систему в безопасное состояние.
  + В HMI предусмотрен индикация: мигать индикатор набора воды.
  + Программе можно разрешить, например, режим **набор воды**: оператор нажимает кнопку "Набор воды" (футер, элемент 7), которая, вероятно, открывает клапан до тех пор, пока уровень не появится. Этот функционал – вспомогательный, но можно реализовать: при ALM\_NoWater=true и нажатии "Fill Jacket", открыть клапан GVS и/или насос циркуляции для заполнения рубашки. Однако подробности это вне основной задачи, упомянем только.
  + Сброс аварии происходит автоматически, как только датчик уровня увидит воду (но мы можем оставить бит 12 установленным до ручного сброса, чтобы зафиксировать событие, хотя индикатор на панели гаснет сразу при появлении воды).
* **Bit 13 – Преждевременный останов пастеризации.** Устанавливается, если цикл пастеризации был прерван до штатного завершения (не по аварии питания). Причины: оператор нажал *Стоп* или авария мешалки/ТЭНа и проч.
  + Менеджер режимов при вручную останове пастеризации (CMD\_StopPasteur) помечает эту ситуацию: ALM\_PasteurAbort=1.
  + Аналогично, если другая авария привела к останову, этот бит тоже можно выставить, чтобы в журнале отразить "не успешно завершено".
  + Этот бит носит **информационный характер**; он не блокирует запуск режимов напрямую, но позволяет панель/SCADA оповестить о том, что последняя пастеризация сорвана. Оператор после устранения может сбросить бит (например, кнопкой "Сброс аварий").
* **Bit 14 – Преждевременный останов мойки.** Аналогично предыдущему, но для CIP. Ставится, если мойка не прошла все этапы (например, нажали Стоп CIP на 3 этапе или авария).
* **Bit 15 – Обрыв связи с модулем расширения ПРМ.** В исходной системе ВДП, для CIP-режима требовалось подключать расширение (выходы PRM). Бит 15 сигнализирует потерю связи с этим модулем. В нашей конфигурации ПР102 встроенно содержит 14 DO (эквивалентно PRM), поэтому отдельного модуля может не быть. Однако если вдруг используется внешний модуль (например, аналоговый модуль для доп. датчиков), контроллер может выдавать статус о его отсутствии. Мы этот бит в текущем проекте считаем **резервным** или используем под "нет связи с панелью", как обсуждалось, – решено оставить как **TBD**. По умолчанию =0, при ошибке связи с каким-либо устройством расширения – =1.
  + Если ПР102 без внешних модулей, этот бит никогда не должен срабатывать, иначе его можно игнорировать или удалить из маски.

Модуль аварий реализует описанные правила:

* Постоянно мониторит критичные входы (DI\_MixerFault, DI\_HeaterFault, DI\_LevelJacket, а также состояние analog input error флагов).
* Обрабатывает внутренние сигналы (флаги прерывания, PID ошибки).
* При необходимости останавливает процессы через взаимодействие с Mode Manager и Actuator Manager:
  + Например, как только ALM\_NoWater=1, аварийный модуль может послать менеджеру режимов команду на немедленный стоп текущего режима (если был нагрев). Mode Manager тогда остановит пастеризацию, CIP или ручной нагрев.
  + Но большую часть таких действий выполняют сами модули, "узнав" об аварии (мы глобальные флаги сразу всем делаем доступными). Поэтому можно без дублирования: Actuator Manager сам читает ALM-флаги и реагирует, Mode Manager – тоже (особенно ALM\_PowerFail – сигнал не продолжать цикл).
* Формирование журнала аварий: модуль аварий при каждом возникновении нового аварийного события записывает запись в **журнал аварий** (см. раздел журналов). В записи фиксируется код аварии (номер бита) и метка времени RTC. Журнал хранится в кольцевом буфере (Retain массив записей). Панель или SCADA могут отображать этот журнал для анализа. Например, при обрыве датчика будет запись "время ХХ:ХХ – Обрыв датчика продукта". Коды и расшифровки соответствуют битам маски.
* Сброс аварий: через HMI предусмотрена кнопка "Сброс" аварий. Она может либо писать 0 в регистр 500 (что мы не позволим, Read-only), либо устанавливать отдельную команду CMD\_ResetAlarms. Модуль аварий, получив такой сигнал, очищает все сбрасываемые флаги, **кроме** тех, которые ещё актуальны. Например, если датчик всё ещё неисправен, сброс не снимет bit0 – он сразу снова выставится. А вот бит13/14 (останов) можно сбросить. После сброса соответствующие индикаторы на панели гаснут.

**Интерфейсы:**

* **Входы:** аппаратные DI (аварийные), данные от других модулей (сигналы ошибок, статусы).
* **Выходы:** глобальная ALM\_Mask (Word) для SCADA, отдельные boolean ALM\_\* для внутренних нужд (они же биты этой маски). Также, как упоминалось, взаимодействие: может дергать CMD\_StopPasteur, но мы избегаем циклических зависимостей – проще, что Mode Manager сам смотрит ALM\_Mask и принимает решение.
* **RTC-время:** модуль аварий запрашивает у системных часов текущее время для логирования. ПР102 имеет часы реального времени, OwenLogic позволяет получить time/date.

**Параметры и настройки**

В программе предусмотрен блок хранения параметров (Settings Manager), который инициализирует и хранит технологические настройки, калибровки и др. Пользовательские настройки включают:

* **Уставки пастеризации:** SP\_HeatTemp, SP\_HeatTime, SP\_SmoothTemp, SP\_SmoothDiff, SP\_HoldTime, SP\_HoldDiff, SP\_CoolTemp. Они хранятся как Retain (чтобы после перезагрузки остались прежние). Значения по умолчанию заданы согласно технологическому регламенту. Например, заводские установки: температура пастеризации 68,0 °C, время нагрева 0 мин (не ограничено), плавный нагрев после 50,0 °C, дифференциал 2,0 °C, выдержка 600 с (10 мин), дифференциал на выдержке 1,0 °C, температура охлаждения 27,0 °C. Эти значения будут загружены при первом старте программы (если Retain память пуста) или при сбросе на заводские установки.
* **Параметры нагрева:**
  + PAR\_UseSteam, PAR\_UsePID, PAR\_HeaterGroup2Enabled, PAR\_HeaterGroup3Enabled – булевые опции конфигурации оборудования (см. список опций 11–14). Они влияют на логику регулятора нагрева (как описано). Эти флаги могут быть установлены на этапе конфигурирования проекта (например, Variation #1: электрический нагрев, 3 группы, PID на 1-й – значит PID=true, G2=true, G3=true, Steam=false; Variation #2: электрический нагрев, но без PID – PID=false; Variation #3: паровой нагрев – Steam=true, G2/3 false).
  + PAR\_PID\_Kp, PAR\_PID\_Ti, PAR\_PID\_Td – коэффициенты регулятора, Retain. Задаются опытным путем или через автонастройку.
  + PAR\_HeaterPower% – относительная мощность нагревателей, % (рег.517). По документации, это значение может вводиться, скорее всего, как справочное (в примере 10,12%). Вероятно, оператор вводит общую подключенную мощность нагревателей, а программа использует ее, например, чтобы оценивать время нагрева (через удельную теплоёмкость). В нашем проекте пока не используем эту переменную в расчетах регулятора, но сохраняем ее и отображаем на SCADA.
* **Настройки мешалки:**
  + PAR\_MaxMixerSpeed (% от 50 Гц, 0.1%) – ограничение максимальной скорости вращения (опция 10). Если, например, мешалка конструктивно не должна превышать 70% от номинала, оператор вводит 700, и VFD Manager будет ограничивать уставку. По умолчанию = 1000 (100%).
  + PAR\_UseVFD – булево, использовать ли регулирование частоты (опция 9). Если False, мешалка будет включаться/выключаться на фикс. скорости (50 Гц). Влияет на UI панели (показывать ли слайдер частоты) и на логику VFD Manager (не посылать частоту).
  + PAR\_UseMixerReverse – булево, разрешен ли реверс (опция 8). Если False, кнопка реверса в интерфейсе неактивна, а DO4 не используется.
* **Настройки розлива:**
  + PAR\_DispenseEnabled – включение режима розлива вообще (опция 6). Если False, на главном меню панели режим розлива не отображается. В нашем проекте можно отключать этот функционал, если устройство не оснащено насосом розлива.
  + PAR\_DosingEnabled – разрешение дозирующего розлива (опция 5). Если False, то даже при включенном режиме розлива доступен только непрерывный режим. По умолчанию, вероятно, True, но может выключаться, если нет внешней кнопки или система дозирования не требуется.
  + SP\_DoseVolume – величина дозы для порционного розлива. Это значение вводит оператор (панель в режиме розлива, элемент 4). Единицы – **условные**, т.к. без датчика расхода доза рассчитывается по времени. Вероятно, значение задается в миллилитрах или литрах, а панель/ПЛК пересчитывают во время работы насоса. Возможна процедура калибровки: зеленая кнопка "Пуск" на экране настроек розлива запускает непрерывный розлив для замера – оператор измеряет объем, панель рассчитывает ml/sec. Для простоты, предположим, SP\_DoseVolume непосредственно хранит время работы насоса (в сотых секунды, например). Но лучше хранить в ml, а иметь расчетную переменную. В нашем проекте введем PAR\_PumpFlowRate (Retain, мл/с), чтобы рассчитать время дозы = SP\_DoseVolume / PumpFlowRate. Если калибровка не проводилась, PumpFlowRate можно задать примерно (например, 500 мл/с).
  + PAR\_UseGunSensor – булево, использовать ли датчик положения пистолета (опция 4). Если True, исполнительный модуль учитывает DI\_GunSensor: насос розлива отключается, когда пистолет на месте (датчик сработал), и включается, когда пистолет поднят (если режим розлива активирован). Это позволяет автоматизировать старт/стоп потока по действию оператора.
  + PAR\_UsePortionButton – булево, использовать ли внешнюю кнопку порции (опция 5). Если True, то алгоритм розлива модифицируется: непрерывный и дозированный розлив работают **только при нажатии этой кнопки**. То есть внешняя кнопка становится своего рода "разрешением подачи" (dead-man switch): оператор удерживает ее – насос подает (в непрерывном режиме) или дозированно работает; отпустил – останавливается/не начинает новую порцию. Реализуем: DI\_NextPortion (например) считывается исполнительным модулем:
    - В непрерывном режиме: DO\_PumpDisp включен только пока DI\_NextPortion=1 *и* активирован режим (CMD\_StartDispCont был подан). Если кнопку отпустили – сразу останов.
    - В дозированном: нажатие кнопки действует аналогично нажатию "Пуск дозы" на панели: одна порция начинается при нажатии, если кнопка держится – дозу выдаст полностью; если отпустит раньше времени – мы можем прервать подачу (будет недолив, но логика говорит "только при нажатии"). Скорее всего, от оператора ожидалось держать кнопку всю дозу. После дозы для следующей нужно отпустить и снова нажать (то есть кнопка работает как пуск каждых порций).
* **Настройки мойки:**
  + PAR\_WashEnabled – булево, наличие режима мойки (опция 6). Если False, режим мойки не доступен.
  + SP\_WashCycleTime\_\* – времена для этапов CIP. В руководстве упомянуто, что график работы CIP задан (см. раздел *6.1…6.4*). Вероятно, в сервисном меню есть параметры длительности каждого этапа (например, время ополаскивания, время циркуляции щелочи и т.п.). Список таких параметров можно вывести:
    - Ополаскивание водой (этапы 1,3,5 полной мойки) – длительность, температуру (если нагрев от бойлера, температура определяется GVS).
    - Мойка щелочью – длительность циркуляции щелочного раствора.
    - Мойка кислотой – длительность.
    - Возможно, задержки на дренирование.
    - Температура моющих растворов: у нас GVS (горячая вода) используется, значит поддержание температуры осуществляет сама подача GVS.
    - Мы ограничимся основными: SP\_WashRinseTime, SP\_WashAlkTime, SP\_WashAcidTime (для полного цикла, а для укороченных CIP менеджер просто часть не использует).
  + SP\_WashRinseCount – количество циклов ополаскивания (например, 2 промывки). Но судя по графику, количества фиксированы (полная мойка = 2 ополаскивания воды, по одному до и после каждого химиката). Вариативности нет, поэтому не нужен параметр.
  + PAR\_WashMixerOn – вкл/выкл мешалку при мойке (обычно True, чтобы было перемешивание раствора).
  + PAR\_WashUsePumpCirc – использовать ли циркуляционный насос рубашки во время мойки. Может потребоваться для прокачки системы (например, если ТЭНы используют ту же воду – но у нас, похоже, нет, т.к. нагрев моек идет через GVS). Вероятно, не нужен.
* **Другие настройки:**
  + PAR\_UseProductSensor – использование датчика наличия продукта (опция 16). Если есть DI, который показывает, что продукт налит (например, датчик уровня молока), можно задействовать: не начинать пастеризацию, если продукта нет, или сигнализировать это. Панель имеет индикатор уровня продукта в футере. В нашем проекте: если PAR\_UseProductSensor=True, DI\_ProductLevel (сигнализатор наличия продукта) будет отображаться на индикаторе панели. Программно же пастеризация без продукта – бессмысленна, но и не вредна, кроме холостой работы. Можно блокировать старт пастеризации, если нет продукта (выдавать сообщение). Мы это можем учесть: Mode Manager при команде StartPasteur проверяет DI\_ProductLevel (если включена опция) – если 0, то отвергает команду (не стартует, а устанавливает, например, ALM\_NoProduct (которого нет в маске 500, но можно задействовать резерв bit8). Однако, так как бит 16 не указан в маске, эту аварию, если хотим, нужно вывести на другой уровень (может, только индикатором на панели). Можно просто предотвратить запуск без сигнала, без фиксации аварии.
  + PAR\_AutoStartTime – время автозапуска пастеризации (часы:минуты). Руководство упоминает "Автозапуск пастеризации". Оператор может задать время, к которому пастеризация должна начаться автоматически. Например, с вечера установить 05:00 – утром молоко уже пастеризовано. Реализовать: модуль RTC/таймеров (ниже) ежедневно сверяет текущее время с PAR\_AutoStartTime и если совпало (и режим автозапуска включен, PAR\_AutoStartEnabled), то инициирует CMD\_StartPasteur. В проекте это можно предусмотреть как опцию. (В маске статуса системы бит0 определён как "ручной (поэтапный) режим", а битов для "автозапуск" нет – видимо, это internal).
  + PAR\_BeepOnAlarm – включение зуммера при аварии (панель сама мигает индикатором 4 и управляет своим зуммером, но контроллер тоже мог бы задействовать выход для звука, если был бы – у нас нет, но если панель, то не надо).
  + PAR\_RtcSync – нужно ли синхронизировать часы (не используется, если SCADA делает).
  + **Retain:** Все вышеперечисленные параметры хранятся с признаком Retain, чтобы при перезапуске настройки сохранялись. Для защиты от инициализации с нуля, Settings Manager проверяет специальный флаг STAT\_FirstStart (в Retain): если он отсутствует, значит память была сброшена – тогда записываются дефолты во все SP/PAR, и STAT\_FirstStart ставится, чтобы далее такого не повторялось.

**Интерфейсы:**

* От **SCADA/HMI:** изменение настроек может идти через Modbus (как сказано, 510–517 могут писать SCADA). Также, в панели, вероятно, реализовано меню "Настройки нагрева", "Настройки мойки" и "Сервисное меню", где при изменении параметров происходят записи в PLC.
  + Мы поэтому связываем параметры пастеризации (510–517) и, если нужно, добавили бы CIP-параметры (но они не были указаны, возможно, панель не отдает их SCADA, оставляя на локальном хранении, или просто не предусмотрено менять CIP-времена через SCADA).
  + Прочие опции (булевы включения режимов, выбор датчиков) скорее всего меняются в сервисном меню панели и не нужны SCADA, поэтому им Modbus-адреса не даем (они **только локальные Retain**).
* К **другим модулям:** Settings Manager на старте загружает значения Retain. Все модули (Temp Manager, Mode Manager и др.) напрямую читают соответствующие глобальные переменные SP\_... и PAR\_... при своей работе. Например, Temp Manager берет SP\_HeatTemp как цель нагрева, CIP Manager берет SP\_WashAlkTime как продолжительность этапа и т.д. Также, Settings Manager может предоставлять интерфейсы для изменения – например, Mode Manager по запросу "сброс на заводские" вызывает функцию Settings Manager, который перезаписывает все SP/PAR дефолтами.
* **Retain Storage:** Объём Retain PR102 – ~1016 байт, чего достаточно для десятков переменных. Мы пометим Retain все нужные, следим, чтобы не превышать.

**Журналы и статусы**

Для удобства оператора и интеграции в SCADA, программа ведёт **журналы** основных событий:

* **Журнал аварий:** Каждое возникновение/исчезновение аварийного флага записывается с меткой времени. Формат записи: [Время] [Код/описание события] – например, "12:45:20 Авария: обрыв датчика продукта". В памяти можно хранить фиксированный массив структур, например 50 записей, и индекс текущей. SCADA по Modbus может читать этот массив (например, по отдельному запросу, которого пока нет) или журнал просматривается на панели (в меню "Журнал аварий"). В нашем отчете достаточно, что механизм есть – 50 записей Retain, циклический буфер.
* **Журнал событий:** Отмечаются технологические события: *начало/окончание режимов, переходы этапов*. Например: "13:00:00 Начат цикл пастеризации", "13:30:00 Завершена пастеризация успешно", либо "13:20:00 Пропущен этап нагрева" (если оператор нажал *Пропуск этапа*). Журнал событий (Events) помогает отследить ход процесса. В руководстве упомянут раздел 8.2.2 Журнал событий. Формируем аналогично журналу аварий, кодируем разные события (пастеризация старт/стоп, CIP старт/стоп, ручной режим вкл/выкл, и др.).
* **Журнал запусков (стартов):** Возможно, имеется в виду статистика циклов – сколько циклов пастеризации проведено, даты последних запусков. В ручном отладочном смысле можно комбинировать с журналом событий. Но раз выделено отдельно 8.2.3, сделаем: сохранять дату/время последних N запусков пастеризации и мойки.
* **Журнал дозаций:** (8.2.4) – хранит записи о каждом порционном розливе: время дозирования, какой объем отпущен. Это может помочь вести учет продукта. Реализуем: при каждом выполнении дозы (когда насос отключился по достижении дозы) пишем запись: "[Время] Доза X мл выдана". Можно также журналировать непрерывный розлив (начало/конец и объем если известен).
* **Архив температур:** Также упомянут график температуры продукта и рубашки. В панели может быть реализован график – панель могла сама строить, либо требовать сохранение исторических точек. Мы можем реализовать буфер значений PV\_TempProduct/PV\_TempJacket каждые N секунд. Но это достаточно большой объем данных – возможно, панель сама логирует. В PLC можно хранить разве что сокращенную историю: например, записывать температуру раз в минуту (600 значений за 10 часов, приемлемо). Этот функционал необязателен для управления и может быть вынесен в SCADA. Отметим как потенциальный: **тренд температур** – дополнительный буфер, если потребуется.

Журналы хранятся в Retain (чтобы при перезапуске не терять историю, хотя при полном обесточивании Retain сохранится, а вот сами логи можно хранить и в RAM, если критично по объему). Для надежности, лучше Retain, но осторожно к объему: 50 записей аварий, по 8 байт (время + код) – 400 байт, терпимо.

**Интерфейсы:**

* Заполнение журналов происходит от соответствующих модулей:
  + Аварийный – пишет в журнал аварий.
  + Mode Manager – пишет события "режим начат/остановлен".
  + Temp/CIP Manager – пишет "этап переключен", "этап завершен".
  + Dispense Manager (можно выделить подфункцию) – пишет данные о дозах.
* Чтение журналов: Панель HMI имеет отдельные экраны журналов, которые скорее всего сами запрашивают записи у контроллера. Можно реализовать это как массивы INT (например, массив [50] для кодов и [50] для меток времени). Поскольку формат времени удобнее ЧЧ:ММ, можно хранить например UNIX-время (Epoch) 32-bit на запись, или два слова (дата и время в BCD). Weintek умеет обрабатывать, если настроить. Тут возможны сложности, можно упростить: хранить строки ASCII – но Modbus этого не передаст эффективно. Вероятно, в реальном проекте на панели Weintek эти журналы реализованы самим HMI (он хранит записи). Но нам нужно хотя бы концептуально описать.
* Для краткости: **в рамках контроллера** мы введем массивы:
  + LogAlarms[50] – массив структур с полями (Time, AlarmCode).
  + LogEvents[100] – (Time, EventCode, maybe Value).
  + LogDoses[100] – (Time, VolumeIssued).
  + Эти массивы можно отобразить через сетевые переменные, но лучше – панель считывает все записи скриптом. Detailing not needed here.

**RTC, таймеры и отложенные действия**

Контроллер ПР102 оснащен часами реального времени (RTC), которые используются:

* Для отметки времени в журналах (как описано).
* Для **автоматического таймера**: Панель имеет экран "Таймер" – по описанию, это кухонный таймер, не связанный напрямую с пастеризацией (например, можно запустить отсчет 1 час – и по окончании пищалка). Этот таймер не влияет на управление, но должен работать параллельно. Мы можем реализовать его внутри контроллера:
  + Переменные TimerSetTime (часы:мин:сек) и TimerRemaining и флаг TimerRunning. Когда панель пишет TimerSetTime и команду старт, Timer Manager сохраняет текущее RTC, рассчитывает TargetTime = Now + SetTime и начинает отслеживать. Пока Now < TargetTime, горит индикатор и идет обратный отсчет (TimerRemaining = TargetTime - Now). По истечении – устанавливает флаг завершения и запускает звуковой сигнал (в панели).
  + Однако, возможно, панель Weintek имеет встроенный элемент таймера, который сама умеет считать – но судя по описанию, реализовано на PLC, т.к. по истечении времени PLC должен открыть окно "Таймер завершил работу" и зуммер. Вероятно, PР102 посылает на панель команду (например, sets some bit or registers) по окончанию.
  + В нашем проекте, Timer – вспомогательная функция, но можем прописать: *RTC/Timer Manager* осуществляет эту функцию, используя системный SysTime OwenLogic.
* **Автозапуск процессов:** Если введен параметр PAR\_AutoStartTime для пастеризации, Timer Manager ежедневно сравнивает RTC время с этим параметром:
  + Если часы совпали и в этот день цикл еще не запускался (нужно флаг, чтобы один раз в день запуск), он дает команду CMD\_StartPasteur автоматически. Это реализует опцию автозапуска пастеризации (опция 3.1).
  + Можно также добавить PAR\_AutoStartCIP (например, по расписанию мойку в ночное время).
  + Эти функции опциональны; если PAR\_AutoStartEnabled = False, они игнорируются.
* **Отложенные действия:**
  + *Задержка перед повторным запуском:* после окончания пастеризации может потребоваться блокировка мгновенного повторного старта (дать оборудованию остыть или оператору очистить). Можно установить таймер (например, 5 мин) на повторный старт.
  + *Продолжительность этапов:* В пастеризации этап выдержки длится SP\_HoldTime – Timer Manager (или Temp Manager) запускает таймер отсчета секунд. CIP Manager стартует таймеры для каждого этапа мойки. Все эти таймеры лучше делать с помощью стандартных таймерных блоков (TON/TOF) либо отслеживать RTC дельты.
  + *Пропуск этапа нагрева:* Если SP\_HeatTime > 0, по истечении заданных минут с начала нагрева Timer Manager сигнализирует Temp Manager о тайм-ауте нагрева. Тот может либо автоматически перейти к выдержке (если предусмотрено), либо отобразить кнопку "Пропустить этап 1" (как указано: кнопка пропуска появляется, если время нагрева истекло). Вероятно, система не переходит автоматически, а ждет решения оператора (вдруг продукт не нагрелся, он может решить подождать больше или прекратить). Мы можем: если нагрев не достиг SP за SP\_HeatTime, ставим флаг STAT\_HeatTimeout = 1, который панель использует, чтобы показать кнопку "Пропустить этап". Если оператор нажал её (CMD\_SkipHeatStage), Mode Manager прекращает этап нагрева и форсирует переход к выдержке, выставляя PV\_TempProduct = SP (или просто без этого, начнет выдержку при текущей температуре). Одновременно ALM\_PasteurAbort можно поставить или специальный флаг "нагрев не достигнут" (в маске не было отдельного).
  + *Отсчёт времени до конца мойки/этапа:* Регистры 507–509 обновляются Timer/CIP Manager-ом:
    - MB\_TimeModeDisplay (507) – выбор отображения: например, оператор на панели переключает, хочет видеть "сколько осталось до конца мойки" vs "до конца текущего этапа" vs "сколько прошло". Панель запишет соответствующий код 0/1/2. CIP Manager, зная текущую фазу и её таймеры, заполнит MB\_WashTimeMin/Sec (508/509) тем или иным значением. Например, если MB\_TimeModeDisplay=0 – подсчитать, сколько минут и секунд осталось до завершения всей программы мойки (сумма оставшегося времени текущего и следующих этапов). Если =1 – время до конца текущего этапа (по его таймеру). Если =2 – сколько прошло с начала мойки (разница старт-время). Эта функциональность помогает оператору ориентироваться во времени. Мы её учитываем: CIP Manager умеет рассчитывать эти показатели и записывать каждые секунды в 508/509.
* **Системные напоминания:** Может быть, по RTC можно ввести напоминания (например, обслужить насос через 100 часов наработки, или сигнализировать, что прошло 24 часа с пастеризации). Пока не требовалось.

**Интерфейсы:**

* Используются системные функции RTC – OwenLogic предоставляет, например, функцию чтения текущего времени. Timer Manager должен быть циклическим (каждую секунду хотя бы).
* CMD\_TimerStart, CMD\_TimerStop – команды панели на кухонный таймер, ссылаются на Timer Manager.
* Timer Manager выставляет STAT\_TimerDone когда его таймер доходит до 0, панель на это реагирует (открывает окно, пищит).
* AutoStart uses CMD\_StartPasteur as output at right time.

На этом описание модулей завершается. Все модули взаимодействуют по заранее описанным глобальным переменным интерфейса; их совокупность обеспечивает устойчивое и понятное управление процессом. Ниже представлены диаграммы состояний для режимов и таблицы переменных, отражающие распределение функций.

**Диаграммы состояний и переходов**

Для каждого основного режима разработаны диаграммы состояний (state charts), определяющие возможные этапы и условия переходов. Ниже приводятся описания этих диаграмм и таблицы переходов.

**Автоматическая пастеризация (цикл)**

**Состояния процесса пастеризации:**

1. **Ожидание запуска (Idle):** Режим пастеризации не активен. Находится в фоне, поддерживаются другие режимы. Переход в **Нагрев (Stage 1)** происходит при получении команды CMD\_StartPasteur (например, оператор нажал "Пуск" на панели). Условия входа: продукт должен быть залит (если датчик продукта используется), рубашка с водой (датчик уровня воды в норме), нет активных аварий, режимы CIP/ручной/розлив не выполняются. При старте менеджер режимов устанавливает бит0 регистра 504 в 1 (этап 1 активен), а бит "пастеризация завершена" (5) = 0.
2. **Этап 1 – Нагрев продукта:** В этом состоянии система включает нагреватели и начинает поднимать температуру продукта к заданной SP\_HeatTemp. Мешалка включена (непрерывно или циклически). **Условия выхода:**
   * Основное условие: PV\_TempProduct ≥ SP\_HeatTemp (продукт достиг требуемой температуры). Тогда автоматически осуществляется переход к **Этап 2 (Выдержка)**. Бит0 маски этапов 504 сбрасывается, бит1 устанавливается.
   * Дополнительное условие (опционально): истечение времени нагрева SP\_HeatTime (если оно > 0). Если задан ограничитель времени нагрева и он вышел, а температура еще не достигнута, появляется возможность перехода по таймауту. Система либо:
     + Автоматически переходит к выдержке (что означает пастеризация начнётся при недогретом продукте – обычно неприемлемо без вмешательства).
     + Либо, как реализовано в UI, предлагает оператору **пропустить этап**. На панели появляется кнопка "Пропуск этапа №1". Оператор по своему усмотрению нажимает её (указывая, что согласен прервать нагрев досрочно и перейти к следующему этапу). Если нажата CMD\_SkipHeatStage, переходим в Этап 2, но фиксируем ситуацию как **преждевременный переход**: устанавливаем бит13 "останов пастеризации" (или отдельный флаг) чтобы указать, что цикл нарушен.
     + Если оператор не пропустил и, например, добавил время или поднял уставку, можно продолжать нагрев. (В нашем алгоритме можно задать SP\_HeatTime=0 чтобы отключить таймаут и кнопка не появлялась).
   * Аварийные выходы: если во время нагрева случается авария (например, ALM\_NoWaterJacket), процесс немедленно переходит в **Аварийное завершение** (см. ниже). Если нажата CMD\_StopPasteur вручную – переход в **Досрочное завершение**.
3. **Этап 2 – Выдержка продукта:** Продукт находится на целевой температуре. Таймер выдержки SP\_HoldTime запускается (например, 600 с). Весь период поддерживается температура: регулятор подогревает по необходимости, мешалка продолжает работать. **Условия выхода:**
   * Истечение времени выдержки. Когда таймер достиг 0, считается, что продукт пастеризован требуемое время. Менеджер режимов переводит процесс в **Этап 3 (Охлаждение)**. Бит1 маски 504 =0, бит2 =1. Одновременно сбрасывается обратный отсчет на панели.
   * Ручное прерывание: если оператор нажал *Стоп*, либо случилась авария, происходит переход в аварийное или досрочное завершение (описано ниже).
   * (Опция) Пропуск выдержки: теоретически, если время выдержки 0 (некоторые процессы типа высокотемпературной пастеризации подразумевают мгновенный переход к охлаждению), этап 2 будет пропущен автоматически. В этом случае программа сразу после нагрева перейдет на охлаждение. Бит1 может не успеть устанавливаться – но мы все равно можем отобразить, что этап2 был "нулевой длительности".
4. **Этап 3 – Охлаждение продукта:** Открывается клапан холодной воды (DO\_ValveCold), включается циркуляционный насос охлаждающей воды (если используется; у нас, предположительно, рубашка с водой охлаждается проточной водой/льдом). Продукт охлаждается от температуры пастеризации до SP\_CoolTemp. Мешалка обычно включена (для теплообмена). **Условия выхода:**
   * PV\_TempProduct ≤ SP\_CoolTemp. Требуемая температура охлаждения достигнута (например, 27,0 °C). Тогда:
     + Закрывается клапан охлаждения (Cool\_Demand = 0).
     + Процесс переходит в **Этап 4 – Выравнивание температур**. Бит2 маски 504 =0, бит3 =1.
   * Если есть требование **не охлаждать** (например, пастеризация высокотемпературная без охлаждения – не наш случай), SP\_CoolTemp мог бы быть задан равным температуре окружающей среды, и тогда охлаждение не происходит вовсе. Но обычно SP\_CoolTemp ниже пастеризационной, так что этап обязателен.
   * Аварии/Стоп – аналогично предыдущим этапам, при событии – прерывание.
5. **Этап 4 – Выравнивание температуры:** Пассивный этап. Клапаны закрыты, нагреватели выключены. Продукт и рубашка какое-то время (например, 1–5 минут) остаются без воздействия, чтобы разница температур сократилась. **Условия выхода:**
   * Основное: истечение условного времени выравнивания. Можно не вводить специальный таймер, а считать, что сразу после закрытия охлаждения начинается выравнивание и длится фиксированное малое время или вплоть до команды оператора. В руководстве указано: "если в сервисных настройках пункт 'Хранение после пастеризации' включен, то по завершению этапа 4 система не отключает программу, а продолжает...". Значит, окончание этапа 4 – обычное завершение, если **хранение выключено**. Если хранение включено, то вместо завершения – переход в этап 5.
   * Мы реализуем: PAR\_StoreEnabled (что соответствует пункту меню 8 "Режим хранения после пастеризации: Вкл"). Если False, то как только продукт охладился (этап 4 можно мгновенно завершить, или выдержать 1 минуту для порядка) – пастеризация считается **завершенной**. Менеджер режимов переходит в **Завершено**. В маске 504 устанавливается бит5 "пастеризация завершена", а биты этапов сбрасываются.
   * Если PAR\_StoreEnabled = True, то вместо завершения – переход в **Этап 5 – Хранение**. Бит3 сбрасывается, бит4 "хранение" =1.
   * Оператор также может вручную прервать хранение (нажав стоп), что выведет из этого этапа.
6. **Этап 5 – Хранение продукта:** В этом состоянии система поддерживает продукт при конечной температуре. Как правило, SP\_CoolTemp поддерживается охлаждением: при превышении на +X °C включается клапан подачи охлаждения, при снижении – выключается (хистерезис). Мешалка, вероятно, выключена или периодически включается для перемешивания по таймеру, чтобы продукт оставался гомогенным. **Условия выхода:**
   * Режим хранения продолжается неопределенно долго, пока оператор не остановит программу. На панели, после включения хранения, кнопка "Стоп" остается доступна. Когда оператор считает процесс завершенным (например, готов к выгрузке продукта), он нажимает CMD\_StopPasteur.
   * Также возможен выход по аварии (например, электропитание или др., тогда процесс прерывается).
   * После выхода из хранения происходит **Окончание пастеризации**: все агрегаты выключаются, MODE\_PasteurActive = 0, в рег.504 ставится бит5 (завершено). Цикл завершен штатно (если Stop – впрочем, Stop при хранении это штатное окончание, не аварийное, т.к. хранение могло продолжаться сколь угодно).

**Завершение пастеризации:**

* **Штатное завершение** – либо после этапа 4 (если хранение отключено) либо после этапа 5 (если было хранение). В этом случае процесс считается успешно выполненным. Менеджер режимов сбрасывает режим пастеризации, фиксирует время окончания, инкрементирует счетчик циклов. Бит5 "пастеризация завершена" может быть установлен с момента окончания.
* **Досрочное завершение оператором** – если нажата кнопка "Стоп" на любом этапе (1–5). Система немедленно:
  + Отключает нагрев/охлаждение/мешалку,
  + Переходит в Idle (режим пастеризации неактивен),
  + Ставит флаг *преждевременного останова* (ALM bit13), чтобы отразить, что цикл не доведен.
  + Журнал событий получает запись "цикл прерван пользователем на этапе X".
* **Аварийное прерывание** – при аварии: алгоритм похож на досрочное, только причина – авария. Плюс, бит13 тоже можем ставить (прерывание), и конкретная авария уже отражает причину. Например, при ALM\_NoWaterJacket возникшем на этапе 1: цикл останавливается, bit13=1, ALM12=1. После устранения воды оператор может начать новый цикл заново.

Для наглядности можно представить временную диаграмму пастеризации (см. рис. 2 руководства: индикация этапов) – у нас она детализирована.

**Таблица переходов для пастеризации:**

| **Текущее состояние** | **Условие (событие)** | **Действия при выходе** | **Следующее состояние** |
| --- | --- | --- | --- |
| Ожидание (Idle) | CMD\_StartPasteur получена, нет блокировок | Установить бит0 (этап1); запустить нагреватели, мешалку; STAT\_AutoActive=1 | Этап 1 – Нагрев |
| Этап 1 – Нагрев | PV\_TempProduct >= SP\_HeatTemp | Запомнить достижение уставки (лог); выключить нагрев (если не PID, иначе PID сам); запустить таймер выдержки; бит0→0, бит1→1 | Этап 2 – Выдержка |
| Этап 1 – Нагрев | Время нагрева SP\_HeatTime истекло (и temp < SP) | Поставить STAT\_HeatTimeout=1 (для UI); **ждать** команды оператора или автопереход, если разрешено | (остается Этап 1, но таймер остановлен) |
| Этап 1 – Нагрев (таймаут) | CMD\_SkipHeatStage от оператора | Записать факт неполного нагрева (ALM13 или лог); бит0→0, бит1→1; начать выдержку (даже при недогреве) | Этап 2 – Выдержка (форсировано) |
| Этап 2 – Выдержка | Таймер выдержки истек | Закончить выдержку; бит1→0, бит2→1; открыть клапан охлаждения | Этап 3 – Охлаждение |
| Этап 3 – Охлаждение | PV\_TempProduct <= SP\_CoolTemp | Закрыть клапан охлаждения; зафиксировать достижение охлаждения; бит2→0, бит3→1 | Этап 4 – Выравнивание |
| Этап 4 – Выравнивание | PAR\_StoreEnabled = False (хранение выкл), + (при необходимости выдержана мин. задержка) | Пастеризация завершена; бит3→0, бит5→1; отключить все механизмы; STAT\_AutoActive=0 | Завершено (Idle) |
| Этап 4 – Выравнивание | PAR\_StoreEnabled = True | Перейти в режим хранения; бит3→0, бит4→1; оставить включенным поддержание охлаждением (по необходимости) | Этап 5 – Хранение |
| Этап 5 – Хранение | CMD\_StopPasteur (оператор завершил) | Отключить охлаждение; бит4→0, бит5→1; STAT\_AutoActive=0 | Завершено (Idle) |
| Любой этап 1–5 | Авария (ALM\_x) | Отключить нагрев, охлаждение, мешалку немедленно; bit0-4→0 (все этапы off); ALM маска установлена; ALM13/14 при необходимости; STAT\_AutoActive=0 | Аварийный останов (Idle) |
| Любой этап 1–5 | CMD\_StopPasteur (оператор) | Отключить всё оборудование; bit0-4→0; установить ALM13; STAT\_AutoActive=0 | Досрочный останов (Idle) |

*(Примечание: состояние "Завершено" не длительное – это фактически возвращение к Idle с установленным флагом "завершено". После завершения можно сразу подготовить к следующему циклу.)*

**Ручной режим**

**Состояния ручного режима:** В ручном режиме оператор самостоятельно включает/выключает отдельные механизмы. Нет строгой последовательности шагов, но можно выделить состояние "Активен" vs "Не активен".

* **Выключен (Idle)**: По умолчанию ручной режим неактивен (если он опционален, может быть выключен совсем). Переход в **Активный ручной режим** происходит при нажатии кнопки "Ручной режим" на панели (при условии, что никакой автоцикл не выполняется). Менеджер режимов проверяет: пастеризация, CIP, рецепты должны быть остановлены – если нет, требует подтвердить остановку (в UI, возможно, при входе в ручной предлагают остановить автопроцесс). В нашем случае Mode Manager не позволит войти, пока AutoActive не 0.
* **Ручной режим активен:** Установлен глобальный флаг MODE\_ManualActive, SCADA-регистр 518 бит0 = 1 (индикация ручного режима). На панели открывается экран ручного управления нагревом/охлаждением/мешалкой. В этом состоянии:
  + **Подрежимы (неявные):** оператор может независимо манипулировать:
    - **Мешалкой**: нажатием "Пуск мешалки" (элемент 4.1 на панели). Вероятно, две кнопки – *Включить* и *Отключить*, или один переключатель. При нажатии "Мешалка Вкл" устанавливается переменная CMD\_ManMixerOn = True, менеджер исполнительных механизмов включает выход DO\_MixerFwd. Если предусмотрен реверс, может быть отдельная кнопка "Реверс" – возможно, переключатель направления. Например, 4.1 "Мешалка" – возможно, отобразить ее состояние и выбрать направление (UI подробностей нет). Реализуем:
      * Если реверс опция включена, в ручном режиме можно выбрать направление (можно двумя кнопками: "Вперед" и "Назад", взаимоисключающие). Переменные: CMD\_MixerFwd\_Man, CMD\_MixerRev\_Man (при нажатии одной, отключается другая).
      * Если реверс опции нет, одна кнопка "Мешалка On/Off".
    - **Нагрев №1, Нагрев №2:** (элементы 4.2 и 4.3). Судя по названиям, в ручном режиме раздельно управляются две группы нагревателей:
      * *Нагрев №1* скорее всего соответствует первой группе ТЭНов/тиристору (например, PID-группа). Нажатие "Нагрев1 Вкл" приводит к подаче напряжения на эту группу. Если там PID, то, возможно, ручное включение означает 100% выход PID (тиристор полностью открыт). Либо, если PID активен, могли бы дать оператору задать аналоговое значение вручную, но интерфейс об этом не говорит – вероятно, просто On/Off.
      * *Нагрев №2* – включает вторую группу ТЭНов (и, если 3 группы, возможно, включает сразу обе оставшиеся или одну – но UI не имело "Нагрев №3"). Вероятно, предполагается, что система либо с двумя группами (тогда Нагрев2 = группа2), либо, если 3 группы, то 2 и 3 соединены вместе в ручном режиме (или вообще 3-я не используется в variant1). У нас 3 DO для ТЭНов; будем считать, что "Нагрев №2" включает обе дополнительные группы для упрощения. То есть:
        + CMD\_ManHeat1On управляет DO6 (ТЭН1 или тиристор).
        + CMD\_ManHeat2On управляет DO7 и DO8 (ТЭН2 и ТЭН3) одновременно. (Либо, если 3-я не подключена, только DO7).
      * Если опции G3Disabled, DO8 не будет действовать. Если система Steam (паровой клапан), то, возможно, "Нагрев1" управляет паровым клапаном (открыть/закрыть), а "Нагрев2" вообще скрыт, т.к. электронагрева нет.
      * Менеджер исполнительных механизмов следит: **при включении нагрева вручную обязательно должна быть вода в рубашке**. Если DI\_LevelJacket = 0, он откажет во включении (или сразу ALM\_NoWater).
      * Также, возможно, нужно оператору самому включать циркуляционный насос рубашки. Но UI такой кнопки не показывает – вероятно, при ручном включении нагрева, насос автоматически включается (т.к. иначе ТЭНы в стоячей воде могут локально перегреться). Мы сделаем: когда CMD\_ManHeat1On or 2On = True, исполнительный модуль также включает DO\_PumpCirc (если он существует в конфигурации).
    - **Охлаждение:** (элемент 4.4). Кнопка "Охлаждение" включает клапан холодной воды (DO2 в системе). Нажатие устанавливает CMD\_ManCoolOn = True, исполнительный модуль открывает клапан ХВС (и возможно включает насос циркуляции, если он отдельно нужен для охлаждения тоже; в схеме, видимо, нет отдельного насоса – возможно, вода самотёком? Но скорее нужен тот же насос).
      * При ручном охлаждении, менеджер также должен не допустить одновременный нагрев: если включено охлаждение, а нагрев был включен, можно либо автоматически выключить нагрев (вероятнее), либо разрешить, но тогда они просто будут бороться. Логично: **взаимоисключение** – мы реализуем: менеджер исполнительных механизмов при CMD\_ManCoolOn=1 принудительно сбрасывает DO\_Heater\*.
      * Аналогично, если нагрев включён и оператор нажимает охлаждение, лучше нагрев выключить. И наоборот, если охлаждение активно и нажимают нагрев – можно либо не позволить (игнорировать команду, подать сигнал), либо выключить охлаждение. Надежнее – при попытке противоположного действия – выключить предыдущее. Например: оператор случайно включает и нагрев, и охлаждение – получит оба включены. Мы реализуем: при ManHeatOn и уже ManCoolOn, менеджер не включает Heat или выключает Cool; выберем: команду, пришедшую позже, имеет приоритет, а предыдущую снимаем. Это можно на уровне панели исключить (сделать взаимно-исключающие кнопки), но если нет, делаем в логике.
    - **Прочее:** В ручном режиме на панели могли быть и другие кнопки (например, "Насос мойки" – но скорее CIP запускается отдельным режимом, не через ручной). "Насос розлива" – скорее нет, розлив – отдельный режим. В ручном, вероятно, только мешалка, нагрев, охлаждение.
    - О каких-либо других входах (например, "включить звуковой сигнал" или "ручной слив") не упоминается.
  + **Индикация и защита:** Пока ручной режим активен, автоматически никакие шаги не выполняются, оператор отвечает за безопасное сочетание:
    - Если он включает нагрев без воды – сработает авария (и модуль аварий выведет из ручного? Вероятно, нет, ручной режим может оставаться включён, просто авария не даст нагрев).
    - Если он включает одновременно мешалку вперед и назад (невозможно, т.к. должно быть взаимное выключение).
    - Если включает нагрев и охлаждение – как решили, исполнительный модуль будет выключать конфликтующее.
  + Аварийные ситуации в ручном:
    - ALM\_MixerFault – просто отключит мешалку, но ручной режим остается. Оператор увидит индикатор неисправности, сам выключит мешалку.
    - ALM\_HeaterFault – отключит нагреватели. Оператору индикация, он разберется.
    - ALM\_NoWaterJacket – сразу отключит нагреватели (если были включены) и выдаст сигнал – оператор должен залить воду. Ручной режим при этом может оставаться, чтобы он мог управлять клапаном заполнения (но отдельной кнопки fill нет; у нас refill идет, возможно, через CIP ополаскивание, или он прям нальет).
    - Остальные аварии аналогично.
* **Выход из ручного режима:** осуществляется явно по команде – на панели есть кнопка "Назад/Выход" для возврата в главное меню. При этом Mode Manager выключает MODE\_ManualActive (бит0 статуса системы сбрасывается) и переводит все вручную включенные механизмы в Off (т.е. как если бы оператор их выключил). По сути, как только оператор покинул экран ручного режима, все CMD\_Man\* сбрасываются (панель, возможно, сама снимает их, или PLC при отключении флага режима). Если же он забыл выключить что-то и вышел – лучше, чтобы PLC сама все отключила, иначе получится, что вернулся в авто, а ТЭНы остались включены. Поэтому Mode Manager, при уходе из ручного, вызывает *ResetAllOutputs()* – через исполнительный модуль все DO off.
* **Блокировки:** Пока MODE\_ManualActive=1, менеджер режимов блокирует запуск автоматических режимов (пастеризация, CIP, розлив). На панели они, скорее всего, скрыты (главное меню формируется динамически). Но если SCADA кто-то отправит CMD\_StartPasteur – Mode Manager игнорирует или ставит в очередь: у нас – игнорирует, или может выдать ошибку "Невозможно в ручном режиме".

**Сводка по ручному режиму (таблица состояний):**

| **Состояние** | **Условие входа** | **Действия на входе** | **Действия внутри** | **Условие выхода** | **Действия на выходе** | **Следующее состояние** |
| --- | --- | --- | --- | --- | --- | --- |
| Idle (авторежим) | Нажата кнопка "Ручной режим" на HMI | Если автопроцессы активны – остановить/запретить; MODE\_ManualActive=1; бит0 статуса=1; переключить экран HMI | (не применяется) | - | - | Manual Active |
| Manual Active | (при активном ручном режиме) | - | Оператор воздействует на CMD\_ManMixerOn/Off, CMD\_ManHeat1On/Off, CMD\_ManHeat2On/Off, CMD\_ManCoolOn/Off. Исполнительный модуль выполняет команды с учетом условий: <ul><li>Включение мешалки: немедленно DO\_MixerFwd (или Rev) →1. При переключении направления – пауза 2 с, затем включить обратное.</li><li>Включение нагрев1: если разрешено (вода есть, нагр. не в авар.), → DO6 (и/или AO) =1 (100%). Включается насос циркуляции DO1, если Steam=нет.</li><li>Включение нагрев2: → DO7,DO8 =1 (если имеются). Насос циркуляции тоже включается.</li><li>Включение охлаждения: → DO2=1 (клапан ХВС открыт), DO1 (насос циркуляции) =1, DO6-8 принудительно =0 (нагрев выкл).</li><li>Выключение любого механизма: →соответствующие DO=0. Если все нагреватели выкл – насос циркуляции DO1 можно отключить (если он не нужен отдельно для охлаждения).</li></ul> Аварии: <ul><li>При ALM\_NoWaterJacket: DO6-8=0 (нагрев выкл), вывести предупреждение (панель мигает).</li><li>ALM\_MixerFault: DO3-4=0 (мешалка выкл).</li></ul> | Нажата кнопка "Выход" (возврат в меню) или CMD\_ExitManual (внутренний) | MODE\_ManualActive=0; все CMD\_Man\* команды сбрасываются; все DO отключаются (на всякий случай) | Idle (авторежим) |

*(Замечание: В реальном HMI выход из ручного режима, вероятно, просто переключает экран, но логически мы делаем, что это сопровождается отключением режима.)*

**Мойка (CIP)**

Режим мойки (Cleaning In Place) реализован в нескольких вариантах. Пользователь может выбрать один из циклов:

* **Полная мойка (5 этапов)** – включает последовательность: ополаскивание водой, щелочная мойка, ополаскивание водой, кислотная мойка, ополаскивание водой.
* **Мойка со щёлочью (3 этапа)** – укороченная: ополаскивание водой, щёлочь, ополаскивание водой.
* **Мойка с кислотой (3 этапа)** – ополаскивание водой, кислота, ополаскивание водой.
* **Ополаскивание (1 этап)** – только ополаскивание водой.

Вариант цикла пользователь выбирает, вероятно, на экране "Мойка" (например, выпадающий список или отдельные кнопки). В нашем проекте – предположим, 4 кнопки: "Полная", "Щёлочная", "Кислотная", "Ополаск". Mode Manager принимает команду CMD\_StartCIP с параметром или отдельные CMD\_StartCIP\_Full, CMD\_StartCIP\_Alk, etc. (Реализуем отдельные флаги). По запуске CIP:

* Проверяется, что пастеризация не активна, ручной режим и розлив завершены (все несовместимые режимы не работают).
* Устанавливается MODE\_CIPActive = True (возможно, мы не делаем отдельный флаг, а используем режим code, но можно и флаг).
* Состояние CIP включает**подсостояния** – этапы CIP.

**Общие элементы CIP:**

* В процессе CIP используются:
  + Холодная вода (ХВС) – клапан DO9.
  + Горячая вода (ГВС) – клапан DO10.
  + Клапан подачи щёлочи – DO11 (подключен к баку/насосу щелочного раствора).
  + Клапан подачи кислоты – DO12.
  + Циркуляционный насос CIP – DO13 (для прокачки моющего раствора через систему).
  + Клапан слива – DO14 (открывается чтобы слить содержимое).
  + Мешалка – DO3/4 (для перемешивания раствора внутри бака).
  + Нагрев: обычно при CIP **не задействуются ТЭНы рубашки**, так как для нагрева моющего раствора подается горячая вода GVS прямо внутрь бака (или рубашки). В графиках указано "Этап 1 – ополаскивание тёплой водой" и там оба клапана ХВС и ГВС открывались, видимо, чтобы смешать до нужной температуры. Наша программа должна уметь одновременно открывать ХВС и ГВС, если требуется тёплая (см. график: на этапе 1 коротко открыты оба, затем закрыт ХВС, оставлен ГВС).
    - Видимо, стратегия: сначала пустить холодную, затем горячую, или наоборот, чтобы средняя температура получилась умеренной (например, ~40 °C).
    - Однако точно алгоритм смешивания не уточнён; возможно, предполагается: открыть холодную и горячую вместе на некоторое время для "тёплой воды". В графике этап1: сначала оба клапана открыты (для наполнения тёплой водой), потом ХВС закрывается, ГВС еще продолжает (держит температуру?), потом все выключается и насос включен, после чего слив.
    - Мы можем упростить: будем считать "ополаскивание тёплой водой" – открыть оба клапана на X секунд, либо циклически: чередовать, но лучше – параллельно (на полную мойку, например, 1-2 минуты ополаскивания).
    - На этапе щелочи: открыть GVS (чтобы подать горячую воду) и клапан щёлочи, мешалка, насос. График этап2: GVS включен вначале, потом выключен; щёлочь включена в середине на некоторое время, мешалка и насос работают весь этап, слив открыт почти всё время. Возможно, реально:  
      *Наливается немного горячей воды + щёлочь, циркулирует, потихоньку сливается.*
    - После этап2, этап3 – ополаскивание: GVS, насос, слив – смывают щёлочь.
    - Этап4 (кислота): аналогично этап2, только вместо щёлочи – кислота.
    - Этап5: финальное ополаскивание горячей водой для смыва кислоты.
  + В общем, CIP Manager следует заранее определенной *временной программе* по этапам.

**Логика CIP Manager (по шагам):**

Для каждой конфигурации (полная/частичная мойка) – свой список этапов, но можно написать один алгоритм, который:

* Знает последовательность для текущего режима (например, массив структур: тип\_этапа, длительность).
* Выполняет этапы по порядку.

Опишем полную мойку (5 этапов) подробно:

1. **Этап 1: Ополаскивание теплой водой.**
   * **Длительность:** например, 5 минут (в графике указано 13 мин - вероятно полный цикл 13 мин, этап1 ~2 мин).
   * **Действия в начале:** Открыть клапаны ХВС (DO9) и ГВС (DO10) одновременно, запустить насос CIP (DO13).
   * Через некоторое время, можно закрыть ХВС, оставив ГВС (как в графике: холодная ненадолго для начального заполнения, потом только горячая).
   * Мешалка: в графике для этап1 мешалка выкл, т.е. при ополаскивании просто промывают стенки потоком воды, мешать не обязательно.
   * Слив: здесь интересно – на рис. видно, слив включается под конец этапа (видимо, пока наполняют – слив закрыт, потом чтобы слить – открывают). Да, DO14 Слив - показан выкл, потом в конце вкл. Насос – показан выкл, потом вкл, вероятно:
     + Сначала налили воду (без насоса? возможно, самотёком или через нижний патрубок?), потом включили насос для циркуляции/разбрызгивания (?),
     + Потом открыли слив – слили.
   * Возможно, реальный процесс:
     + Открыть ХВС+ГВС на X секунд (налить ~полный объем).
     + Закрыть оба (бак заполнен водой).
     + Включить циркуляционный насос (гоняет воду по рубашке или форсункам внутри, если есть) - \*хотя "насос" у них, скорее всего, циркулирует по трубам рубашки? Либо имеется "моечная головка" и насос качает из бака).
     + После Y секунд циркуляции – открыть слив (выпустить воду).
     + Насос возможно продолжает гнать, чтобы вытолкнуть всю воду в слив.
     + Закрыть слив.
   * Мы упрощенно: будем управлять параллельно:
     + t=0: DO9, DO10 =1 (начало наполнять), DO13=0 (еще не включаем насос, т.к. нужно воды набрать?), DO14=0 (слив закрыт).
     + t=30с: DO9=0 (хватает холодной), DO10=1 (оставляем горячую, еще 30с).
     + t=60с: DO10=0 (прекратили подачу воды, бак наполнен). Включаем DO13=1 (насос циркулирует).
     + t=120с: Открываем DO14=1 (слив) – вода начинает уходить, но насос продолжает, чтобы промывать при сливе.
     + t=150с: Выключаем DO13 (насос) – чтобы не работать на сухую).
     + t=180с: Закрываем DO14 (слив закрыт, этап окончен).
   * (Конкретные времена можно настроить параметрами).
   * **Переход:** по окончании времени этапа 1, переходим к этапу 2. CIP Manager выключает все, что должно быть off (если что-то еще включено).
   * В Modbus-регистре 503 устанавливается bit0=1 во время этапа1, затем при переходе bit0→0, bit1→1.
2. **Этап 2: Мойка щёлочью.**
   * **Длительность:** напр. 10 мин.
   * **Начало:** открыть клапан ГВС (горячая вода) – для наполнения горячей водой системы. Включить насос CIP (для циркуляции). **Мешалка включена** (в графике этап2 мешалка ВКЛ все время).
   * Открыть клапан щёлочи (DO11) – но не сразу: скорее, сначала немного воды, потом добавить щёлочь. В графике щёлочь: выкл, потом вкл, потом выкл – видимо, подается на каком-то интервале, не постоянно.
   * Алгоритм:
     + t=0: DO10=1 (горячая), DO13=1 (насос), DO3=1 (мешалка), DO14=1 (слив?) – в графике слив показан включен уже с начала этапа2. Может они постоянно сливают небольшую часть, чтобы был ток жидкости.
       - Но если слив открыт, жидкость не задержится – возможно, слив частично открыт, дозированно, или имеется какая-то циркуляция замкнутая? Неясно.
       - Раз слив DO14=ВКЛ на всём этапе, значит, система может быть устроена: из бака жидкость откачивается насосом, проходит по трубам, возвращается в бак или сразу в слив. В CIP обычно делают циркуляцию: из бака моющий раствор берется насосом, через форсунки моет, и снова стекает в тот же бак. Если DO14 (слив) открыт, возможно он перенаправляет поток не в бак, а в сток.
       - Впрочем, "слив" может означать клапан, направляющий выход либо в циркуляцию (закрыт = циркуляция), либо в канализацию (открыт = сливает). Если DO14=ВКЛ, то возможно, контур не замкнут, а идёт сразу в канализацию. Но тогда откуда циркуляция?
       - Возможно, DO14 =0 -> раствор циркулирует, DO14 =1 -> сливает из системы.
       - Глядя на график: этап2 слив ВКЛ почти всё время, но в конце тоже ВКЛ. Странно.
       - Может, DO14=ВКЛ означает "открыт путь возврата раствора в бак CIP"? Не согласуется с названием "слив".
       - Можно предположить:
         * Этап2: берут раствор из отдельного резервуара (бак с щёлочью) с помощью насоса CIP, прогоняют через систему (форсунку в баке, рубашки), и всё сразу сливается (не возвращая в бак).
         * То есть, нет замкнутого цикла – щелочной раствор просто расходуется. Это возможно на малых системах, где не собирают обратно химикаты.
         * Тогда "насос CIP" берет щёлочь из бочки -> через рубашку -> в канализацию.
         * DO10 (горячая) подмешивается, чтобы раствор был горячим.
         * DO14 = "клапан слива" открыт, чтобы всё шло в канализацию.
       - Логично:  
         *Start*: открыть щёлочь, горячую, насос – всё течет в слив.  
         *Stop*: закрыть щёлочь, продолжить промывать горячей водой (этап3).
       - Примем эту модель: CIP без восстановления растворов, все химии одноразовые, сразу сливаются. Тогда do14=1 просто все время, do13 качает.
     + t=30с: DO11=1 (подача щёлочи).
     + t=120с: DO11=0 (прекратить подачу щёлочи, остаток времени этапа – просто горячая вода циркулирует, продолжая смывать остатки).
     + t=Х: DO10, DO13, DO3 продолжают. (Слив открыт постоянно).
     + t=end: завершение этапа – DO10=0 (закрыть горячую), DO13=0 (насос выкл), DO3=0 (мешалка выкл) – хотя мешалку можно оставлять, но лучше отключить на конеце, перед следующим ополаскиванием).
   * **Конец:** переход к этапу3.
   * В регистре 503: bit1=1 во время этапа2, затем 0, bit2=1 для этапа3.
3. **Этап 3: Ополаскивание горячей водой** (после щелочи).
   * По названию, скорее "мойка водой" (в тексте 6.1 график: "этап №3 – мойка водой").
   * Задача: смыть остатки щелочного раствора.
   * **Действия:** Подать горячую воду (GVS) через систему + насос. Слив открыт (чтобы гнать в канализацию).
   * Мешалка можно отключить или включена – в графике этап3 мешалка выкл, значит не обязательно мешать.
   * Алгоритм:
     + t=0: DO10=1, DO13=1, DO14=1 (слить).
     + (если хочется, немного закрыть/открыть, но можно просто постоянно).
     + t=end: DO10=0, DO13=0, DO14=1 (оставляем слив, можно его не закрывать между этапами).
   * Здесь, может, не нужен четкий конец – можно непрерывно перейти к след.этапу (если слив открыт, можно сразу подать кислоту).
   * Но ради согласованности: закрыть/отключить всё, потом откроем снова.
   * Регистр 503: bit2=1 на этом этапе.
4. **Этап 4: Мойка кислотой.**
   * Аналогично этапу2, только с кислотным раствором (DO12).
   * **Действия:**
     + DO10 (горячая) =1, DO13=1, DO3=1, DO14=1.
     + Через 30с: DO12=1 (подача кислоты).
     + Через 120с: DO12=0 (стоп подача кислоты, далее только горячая вода).
     + End: DO10=0, DO13=0, DO3=0.
   * Регистр 503: bit3=1 (этап4 активен).
5. **Этап 5: Ополаскивание горячей водой (после кислоты).**
   * Как этап3, смыть кислоту.
   * DO10=1, DO13=1, DO14=1, мешалка выкл.
   * End: DO10=0, DO13=0, DO14=0 (после финального ополаскивания, закрыть слив? можно оставить открытым, но к концу уже все слито, закрыть чтобы система герметична).
   * Регистр 503: bit4=1 (этап5 активен). После завершения – все биты 0.

**Завершение CIP:**

* После выполнения последнего этапа, CIP Manager останавливает режим мойки:
  + MODE\_CIPActive=0.
  + Устанавливает **бит5 "пастеризация завершена"???** Нет, для CIP отдельного "завершено" бита нет, но можно считать, что когда ни один бит 0-4,5-7,8-10,11 активен, то мойка завершена.
  + Возможно, стоит определить битов не было. На панели, наверное, видят, что CIP режим закончился, т.е. экран возвращается.
* Mode Manager переводит систему в Idle (можно снова пастеризовать или др).

**Преждевременный останов CIP:**

* Если оператор нажмет *Стоп* мойки (или вернется в меню), или случится авария:
  + CIP Manager немедленно прерывает цикл:
    - Все клапаны и насосы выключаются.
    - Биты этапов (503) все сбрасываются.
    - Устанавливается ALM bit14 "преждевременный останов мойки".
    - Если что-то осталось в системе (например, бак частично наполнен раствором, а мы остановились) – оператору придется вручную выполнить ополаскивание, но это вне автоматики.
    - В принципе, лучше дать возможность приостановить и продолжить CIP – но, упрощая, считаем, что "стоп" = abort.

**Частичные циклы CIP:**

* **Щёлочная мойка (3 этапа)**: фактически, этапы: Ополаскивание водой (аналог этап1, но, возможно, тоже теплой), Щёлочь (аналог этап2,3?), Ополаскивание водой (финальное).
  + По битам 503: щёлочная мойка – использует бит5-7.
  + В описании битов:
    - bit5 - щёлочная, этап1 (ополаскивание водой),
    - bit6 - щёлочная, этап2 (мойка щелочью),
    - bit7 - щёлочная, этап3 (ополаскивание водой).
  + То есть, 3 этапа: вода, щёлочь, вода. (Горячая вода, щёлочь, горячая вода).
  + CIP Manager при выборе "щелочная" выполняет только первые 3 этапа из полной (с некоторыми различиями: после щелочного не делает кислоту, сразу переходит к конечному ополаскиванию).
  + В частности: после этап3 (ополаскивания после щелочи) – на этом цикл кончается.
* **Кислотная мойка (3 этапа)**:
  + bit8 - кислотная, этап1 (ополаскивание водой),
  + bit9 - кислотная, этап2 (мойка кислотой),
  + bit10 - кислотная, этап3 (ополаскивание водой).
  + Последовательность: вода, кислота, вода. (По сути, пропускается щелочь – аналогично).
  + Реализация: CIP Manager при выборе "кислотная" может просто выполнить этап1 (полной) + этап4 + этап5. Но лучше всё же: заново, т.к. первой ополаскивание – да, как этап1,
    - затем этап2: подать кислоту (как выше),
    - затем этап3: ополаскивание водой.
* **Ополаскивание (1 этап)**:
  + bit11 - ополаскивание, этап1 (ополаскивание водой).
  + Просто выполняется аналог этап1: (вода, насос, слив).
  + CIP Manager при старте "ополаскивание" сделает 1 этап и завершит.

В реализации CIP Manager можно структурировать:

* Иметь таблицы этапов для каждого режима CIP:
* CIP\_cycles = {
* "Full": [ ("Rinse", t1), ("Alkali", t2), ("Rinse", t3), ("Acid", t4), ("Rinse", t5) ],
* "Alk": [ ("Rinse", ta), ("Alkali", tb), ("Rinse", tc) ],
* "Acid": [ ("Rinse", ta), ("Acid", tb), ("Rinse", tc) ],
* "Rinse": [ ("Rinse", tr) ]
* }
  + Время этапов t1..t5 можно хранить в SP\_Wash\_\* параметрах.
  + CIP Manager, исходя из выбранного режима, пройдет через список и для каждого типа вызовет соответствующую подфункцию:
    - do\_rinse(duration),
    - do\_alkali(duration),
    - do\_acid(duration).
  + Эти функции содержат конкретные включения/выключения выходов, как описано.
* Управление битами Modbus 503:
  + Перед началом каждого этапа CIP Manager устанавливает соответствующий бит (в "Этапы мойки" битовую маску), сбрасывая предыдущие.
  + По завершении последнего этапа – все биты 503 =0.
* Время отображения (регистр 507/508/509):
  + CIP Manager обновляет MB\_TimeMin/MB\_TimeSec каждый цикл:
    - Если MB\_TimeModeDisplay=0 ("До конца мойки"), он суммирует оставшиеся времена всех этапов (вкл текущий) и отдает в 508/509.
    - Если =1 ("До конца этапа"), он берет таймер текущего этапа.
    - Если =2 ("Прошло времени"), он считает от начала режима CIP сколько прошло.
  + Менять 507, скорее, панель может (три варианта).
* Аварийные ситуации:
  + ALM\_NoWaterJacket: для мойки это про рубашку. В CIP, возможно, вода в рубашке не нужна (рубашка не нагревается), но этот датчик может сработать, если рубашка не залита и вдруг нагрев включился. Однако нагрев ТЭНов мы не используем, поэтому ALM\_NoWater можно игнорировать (или masked) в CIP – если система пустая, может, и не надо,
    - Однако, если рубашка была пустая, CIP не трогает ТЭНы, это не критично.
    - Но сам датчик может сигнализировать, панель мигать – можно временно игнорировать (мы не будем ничего греть, так что безопасно).
  + ALM\_MixerFault: если мешалка нужна (в CIP – включаем ее, щелочь/кислота), ее отказ – не критично для безопасности, но качество мойки снизится. Мы можем продолжать CIP, но с флагом аварии – оператор потом узнает, что мешалка не работала. Не останавливаем CIP из-за мешалки.
  + ALM\_PumpCircFault (не у нас).
  + ALM\_HeaterFault: ТЭНы не используются, можно игнорировать.
  + Если связь с PRM (внешний модуль) пропадет – CIP остановится? У нас PRM нет, skip.

**Сводная таблица этапов CIP (полная мойка) с действиями:**

| **Этап** | **Активные выходы (время активации)** | **Примечания переходов** |
| --- | --- | --- |
| Этап 1 (Полная/Щелочная/Кислотная/Ополаск): Ополаскивание водой (теплой) | DO9 ХВС (0–30 с), DO10 ГВС (0–60 с), DO13 насос CIP (60–150 с), DO14 слив (120–180 с). Мешалка OFF. | По окончании этапа: закрыть DO14, убедиться DO9, DO10, DO13 OFF. Далее -> следующий этап или конец. |
| Этап 2 (Полная/Щелочная): Мойка щелочью | DO10 ГВС (все время), DO13 насос (все время), DO3 мешалка (все время), DO11 клапан щёлочи (30–120 с), DO14 слив (все время). | По окончании: выключить DO10, DO13, DO3 (мешалку), DO11 уже off; слив оставить открытым (или закрыть мгновенно). -> Этап ополаскивания. |
| Этап 3 (Полная/Щелочная): Ополаскивание водой после щелочи | DO10 ГВС (все время), DO13 насос (все время), DO14 слив (все время). Мешалка OFF. | Конец этапа: выключить DO10, DO13; -> либо следующий этап (если Полная), либо завершение (если Щелочная). |
| Этап 3 (Кислотная) = Этап 1 (ополаскивание перед кислотой) | (Такие же действия, но биты другие) | Далее -> Этап кислота |
| Этап 4 (Полная): Мойка кислотой | DO10 ГВС, DO13 насос, DO3 мешалка – все время; DO12 клапан кислоты (30–120 с); DO14 слив (все время). | По окончании: выкл DO10, DO13, DO3; -> Этап 5 (ополаск). |
| Этап 5 (Полная) / Этап 3 (Кислотная): Ополаскивание водой после кислоты | DO10 ГВС, DO13 насос, DO14 слив – все время; мешалка OFF. | Конец: выкл DO10, DO13; закрыть DO14; завершение цикла. |

*(В щелочном режиме нет кислоты, в кислотном – нет щелочи. Ополаскивание режим = только Этап 1 по аналогии.)*

**Таблица переходов CIP (для полной мойки, аналогично адаптируется к др):**

| **Состояние CIP (этап)** | **Условие завершения** | **Действия перехода** | **Следующее состояние** |
| --- | --- | --- | --- |
| Idle (CIP не активен) | CMD\_StartCIP\_Full (или Alk/Acid/Rinse) | MODE\_CIPActive=1; установить бит этапа 0/5/8/11 в 1; запустить этап 1 (подпрограмма) | Этап 1 (старт цикла) |
| Этап 1 – Ополаскивание | Время этапа 1 истекло | Выключить все соответствующие DO; сбросить бит этап1; если в программе есть этап2 -> установить бит2 (или нужный следующий бит) =1 и выполнить шаг 2; иначе -> переход к завершению | Этап 2 (или Завершение CIP) |
| Этап 2 – Щелочь | Время этапа 2 истекло | Выключить DO10, DO11; мешалку/насос уже выкл; сбросить бит этап2; установить бит3 =1 (следующий этап) и начать ополаскивание | Этап 3 – Ополаск после щел. |
| Этап 3 – Ополаск (после щел.) | Время истекло | Выключить DO10, DO13; сбросить бит3; если Полная -> бит4=1, начать этап4 (кислота); если Щелочная -> завершение CIP | Этап 4 (кислота) или Завершение |
| Этап 4 – Кислота | Время этапа 4 истекло | Выключить DO10, DO12; мешалку/насос выкл; сбросить бит4; установить бит5=1, начать этап5 (ополаск) | Этап 5 – Ополаск после кисл. |
| Этап 5 – Ополаск (после кисл.) | Время истекло | Выключить DO10, DO13; закрыть DO14; сбросить бит5; | Завершение CIP (Idle) |
| Любой этап (1–5) | CMD\_StopCIP или авария (ALM) | Немедленно выключить все DO (9–14,3, насос, мешалка); сбросить все биты 503; установить ALM14=1 (если стоп до конца) | Преждевременный останов (Idle) |

*(Примечание: для кислотной мойки (3 этапа) и ополаскивания (1 этап) переходы аналогичны соответствующим частям таблицы. Мойка Acid: Idle->этап1->этап2(kis)->этап3->Done. Мойка Rinse: Idle->этап1->Done.)*

**Режим розлива (Dispensing)**

Режим розлива включается после пастеризации, когда продукт готов к разливу по таре. Панель имеет экран "Розлив и дозация", где:

* Кнопка 1 – запуск/стоп **непрерывного розлива** (меняет цвет на желтый "Стоп" при запуске).
* Кнопки 2 и 3 – увеличение/уменьшение порции дозы.
* Поле 4 – текущее значение дозы (можно вручную указать значение).
* Кнопка 6 – запуск/стоп **дозированного розлива** (аналогично, становится "Стоп").
* Кнопка 7 – подача следующей порции.
* Элемент 8 – название режима (можно поменять) – возможно, переключение между непрерывным/дозирующим? Но скорее, нет, 1 и 6 – отдельно.
* Опции:
  + В сервисе:
    - "разрешение режима розлива" (вкл/выкл появление на главном экране) – у нас PAR\_DispenseEnabled.
    - "использование датчика пистолета" (опция 8).
    - "использование кнопки подачи порции" (опция 5).
    - "использование частотника" (опция 10), у нас true, т.к. мешалка = частотник (но, видимо, тоже про насос? Едва ли, чаще насос розлива – просто включ/выключ, скорость не регулируется. Опция 10, скорее всего, про мешалку, а не про розлив).
    - "активация режима дозирования" (скрыт намек: [14†L101-L107] – "Без активации режима дозации, в режиме розлива будет доступен только розлив").
      * Это, вероятно, PAR\_DosingEnabled: если False, элементы 2,3,4,6,7 скрыты, только кнопка 1 (непрерывный).
* **Принцип работы:**
  + **Непрерывный розлив:** При нажатии "Пуск постоянного розлива" (пункт1), насос розлива включается и работает непрерывно, пока оператор не нажмет "Стоп".
    - Т.е. CMD\_StartDispCont→ DO\_PumpDisp=1 непрерывно, CMD\_StopDispCont→DO\_PumpDisp=0.
    - Если опция "внешняя кнопка для подачи порции" включена, то текст говорит: "*если используется внешняя кнопка, то непрерывный и дозирующий розлив будут происходить только при нажатии на эту кнопку*."
      * Это значит, что *даже для непрерывного*, помимо нажатия на панель "Пуск", необходимо, чтобы физическая кнопка (например, педаль, DI\_NextPortion) была удерживаема.
      * Как реализовать:
        + Мы делаем, что DO\_PumpDisp = CMD\_DispContActive AND DI\_PortionButton. То есть, оператор должен держать кнопку всё время, либо, что выглядит неудобно – вероятно, "кнопка для порции" – при непрерывном режиме работает как мертвый человек: нажал и держит = льется, отпустил = останов (это ок).
        + Так и сделаем. Причем, панель, если опция=True, [14†L95-L100] "…будет происходить только при нажатии на эту кнопку." – наверно, панель все равно требует нажать "Пуск", но сама подача не пойдет, пока физическая кнопка тоже не нажата.
        + (В панели, возможно, при опции = True, кнопки "Пуск" на экране не задействуются, а внешний кнопка сама запускает, unclear. Но текст указывает "и постоянный, и дозирующий будут происходить только при нажатии".)
  + **Дозированный розлив:** Перед запуском дозирующего, надо указать порцию (пункт4). После задания дозы, нажатие "Пуск дозирующего розлива" (пункт6) запускает подачу: насос выдаст указанную порцию и встанет на паузу. Затем, для продолжения нужно подать следующую порцию – либо кнопкой "Следующая порция" (п.7), либо внешней кнопкой, если подключена (опция 5).
    - То есть цикл работы:
      * Нажали "Пуск дозирующего" – насос включился и отмерил дозу, затем автоматически остановился.
      * Если нужно еще порции: либо оператор нажимает "Следующая порция" – тогда насос снова включится на один отмеренный цикл, и снова остановится.
      * Это можно повторять.
      * Можно прекратить режим дозирования, нажав "Стоп" (кнопка п.6, которая при запуске стала "Стоп").
    - Внутренне: CMD\_StartDispDose → Mode Manager переходит в состояние "Dose active", CMD\_StopDispDose → выход.
    - Pump control:
      * Мы имеем SP\_DoseVolume (в мл) и PAR\_PumpFlowRate (мл/с) – из них считаем DoseTime = SP\_DoseVolume / PumpFlowRate (в сек).
      * В реализации: при запуске дозы, Dispense Manager включает DO\_PumpDisp=1 и запускает таймер. Когда таймер = DoseTime, он выключает насос.
      * Сигнал "порция выдана" – можно установить, чтобы HMI знала. Панель, похоже, автоматически меняет кнопку "Пуск" -> "Стоп" -> обратно, но, возможно, "Стоп" надо нажать, если хотите прервать порцию.
      * [14†L79-L83]: "После нажатия на пуск дозирующего, насос выдаст указанную порцию и встанет на паузу." – Значит, он сам остановится, кнопку "Стоп" жать не нужно, она просто стала желтой на время порции. По окончании она обратно стала "Пуск"? Или остается "Стоп" для возможности досрочно остановить? В тексте – "при запуске меняется на желтую 'Стоп'.", но когда пауза наступила, вероятно, она опять "Пуск".
      * "Для продолжения – кнопка подачи следующей порции." – значит, основная кнопка (п.6) не требуется нажимать повторно для каждой порции, а именно "Следующая порция" (п.7) используется. Возможно:
        + Пока режим дозирования активен, кнопка (6) остается "Стоп" (чтобы выйти).
        + Каждую новую порцию инициирует либо внешняя кнопка, либо кнопка (7) на экране.
    - Внешняя кнопка (опция 5):
      * Если подключена, можно настроить, что вместо нажатия "Следующая порция" на экране оператор нажимает физическую кнопку.
      * [14†L91-L99]: "Если используется внешняя кнопка для порции, то непрерывный и дозирующий будут происходить только при нажатии этой кнопки." – Это слегка по-другому:
        + для непрерывного – как обсудили,
        + для дозирующего, возможно, означает: даже для первой порции надо держать кнопку? Или что каждая порция начинается кнопкой.
      * Логичнее: в режиме дозирования, если опция=True, насос будет выдавать порцию *только когда внешняя кнопка нажата*. То есть, как "dead man": нажал кнопку -> сразу начало порции, отпустил до завершения – останавливается досрочно.
      * Но "будет происходить только при нажатии" может значить:
        + Panel "Пуск дозирующего" только переводит в standby, а реальное начало (и удержание) – внешней кнопкой.
        + [14†L95-L100]: "…при поднятии пистолета… насос включится." – Это про датчик пистолета, но аналогия: "когда поднят пистолет и включен режим розлива, насос включится". А "при нажатии этой кнопки" – likely foot pedal.
        + Возможно, в интерфейсе, если внешний триггер используется, то после нажатия "Пуск дозиров.", система ждет сигнала от педали для каждой порции (вместо нажимания экранной кнопки).
      * Мы сделаем:
        + Если PAR\_UsePortionButton=True, то Panel "Next portion" (7) может не использоваться – оператор просто нажимает педаль для каждой новой порции.
        + Реализуем: Dispense Manager, увидев, что DoseActive и Pump свободен (остановлен после предыдущей дозы), будет ждать DI\_PortionButton нажать. При обнаружении – запускает порцию.
        + Это соответствует фразе: "*постоянный и дозирующий розлив будет происходить только при нажатии на эту кнопку.*"
      * В любом случае, мы позволим и педаль, и экран "Next portion": если педаль – удобнее, если нет – можно через экран.
    - **Остановка дозирования:** Можно нажать "Стоп" (кнопка6). Тогда, если насос в процессе дозы – он останавливается сразу (получится неполная порция), режим выходит из дозирования. ALM13 (преждевр.останов пастеризации) не применим, можно ALM14 – но то "останов мойки". Мы можем не фиксировать как аварию, а как просто завершение режима розлива (считать, что порция прервана).
    - MODE\_DispActive (с указанием submode continuous/dose).
    - Note: Panel said if dosing mode isn't activated, only continuous is available. Likely PAR\_DosingEnabled controls showing dose controls. If Off, we ignore dose commands.
* **Датчик положения пистолета:** (опция 8) [14†L123-L131].
  + "когда насос отключится, при поднятии пистолета и включенном режиме розлива, насос включится." –
  + Интерпретация:
    - Если пистолет (разливочный) снят с держателя (что детектируется как DI\_GunSensor = false, assuming sensor triggered when gun is in place), и режим непрерывного активирован, то насос автоматически включается (продукт льется). Когда пистолет ставят на место (sensor triggers), насос отключается.
    - То есть, можно управлять потоком просто снимая/ставя пистолет, без нажатия кнопки на панели.
    - "когда насос отключится" – наверное, "когда пистолет снят" = sensor off, "насос включится"; "при срабатывании датчика (пистолет на месте) насос отключается".
    - Already the option 4 said "при срабатывании датчика насос отключается", which covers that.
    - Implementation:
      * In continuous mode: if PAR\_UseGunSensor=True, then tie DO\_PumpDisp = DispContActive AND NOT DI\_GunSensor. So if gun is not in place (sensor off), and mode active, pump flows; if gun in place, pump off.
      * In dosing mode: possibly not automatically triggered by gun. Usually, dosing requires pressing Next portion. But maybe they wanted to allow gun to trigger each portion: "когда пистолет поднят и режим розлива включен, насос включится" – suggests for continuous mode, but could also mean portion triggers on picking up gun (less likely).
      * Possibly simpler: gun sensor used mainly to stop flow if gun is put back unexpectedly, or to allow an easy start for continuous. For dosing, the external portion button covers that scenario.
    - We'll implement primarily for continuous:
      * If continuous running (DispContActive=1), then Pump = (GunSensor==0).
      * For dosing, we can perhaps ensure pump will not run if gun is in place either, to avoid spills (like if portion triggered but gun on holder, maybe should not pump). So do similarly:
        + If portion triggered but sensor shows gun still on holder, either wait until it's lifted or skip. But as portion times are short, maybe require operator to lift first.
        + Simpler: treat gun sensor like a secondary interlock: DO\_PumpDisp in dose mode as well = commanded && NOT DI\_GunSensor. So if operator forgot to lift, the portion will only start when he lifts (if the portion timer didn't start until lift? Might complicate).
        + Another approach: if dose triggered and gun not lifted, we can start pumping anyway, but that likely spills because gun might be at rest (or gun maybe has a valve that closes when in holder, unclear).
        + For safety, implement same as continuous: won't pump until gun removed. If user triggers portion but gun in holder, nothing flows until gun lifted; portion timer, we may need to not count until flow actually start (complicated).
        + Alternatively, require by training that they lift gun, then press portion.
      * Considering complexity, we might restrict: gun sensor usage strongly intended for continuous only. For portion, possibly not needed (foot pedal covers portion).
      * We'll implement consistent logic: PumpOut = PumpCommandActive AND ( (UseGunSensor? (DI\_GunSensor==0) : True) ) AND ( (UsePortionButton? DI\_PortionButton: True) ).
* **Concurrent modes**:
  + While dispensing mode is active, other modes (pasterization, CIP, manual) are locked out. Panel main menu hides others if dispensing mode is turned on (ref. "после включения он появится на главном экране" – suggests toggling in settings).
  + If CIP might be needed after dispensing (for cleaning system), operator should exit dispensing mode first.
  + If inadvertently try to start CIP while dispensing active, Mode Manager will ignore or queue – better to not allow.

**Состояния Dispense:**

* Idle (Dispense mode not active).
* Dispense Continuous active:
  + Substate Off vs Pumping (or simply Pumping is on/off by a flag).
* Dispense Dose active:
  + Substate Idle (waiting to start portion), Pumping (portion in progress), Paused (between portions).
* We might unify as one state "Dispense Active", with submode variable indicates cont or dose:
  + If cont: start pump on command, stop on command or conditions.
  + If dose: handle one portion cycle logic repeatedly.
* But easier for clarity: treat continuous and dosing almost as separate modes, but since they share resources (pump, panel screen), they can't be truly concurrent:
  + Panel distinguishes by separate start buttons. Possibly you can inadvertently press both start cont and start dose? Probably interface disables one if other running. But if not, Mode Manager should handle:
    - If cont mode running and user tries "start dose" – either ignore or stop cont and start dose. But likely UI prevents (maybe hides dose controls if cont running).
    - We'll treat them as mutually exclusive. Mode Manager will only allow one.
* So:
  + MODE\_DispContActive and MODE\_DispDoseActive two bool (or one mode variable with values e.g. 1=cont,2=dose).
  + Starting one stops the other if was active.

**Таблица переходов (dispensing):**

| **Состояние Dispense** | **Условие (событие)** | **Действия** | **Следующее состояние** |
| --- | --- | --- | --- |
| Idle (Dispense off) | CMD\_StartDispCont (непрерывный пуск) | MODE\_DispContActive=1; включить режим на панели (напр., индикатор), подготовить управление насосом | Continuous mode active |
| Idle (Dispense off) | CMD\_StartDispDose (дозир. пуск) | MODE\_DispDoseActive=1; инициализировать дозирующий цикл (сброс счетчика порций, готовность насоса) | Dose mode active |
| Continuous mode active (pump off initially) | CMD\_StartDispCont (если, напр., reused as toggle) | (Если команда используется как toggle, то она уже привела сюда; иначе, pump off ->pump on triggered by separate variable?) **Непосредственно**: сразу включить насос при запуске, *если нет sensor/pedal условий.* См. ниже. | Continuous pumping (pump ON) |
| Continuous pumping (Pump ON) | CMD\_StopDispCont (или cont toggled off) | Отключить насос (DO\_PumpDisp=0); MODE\_DispContActive=0; вывести режим из активного (панель кнопка вернется "Пуск") | Idle (Dispense off) |
| Continuous pumping (Pump ON) | External conditions: DI\_GunSensor triggered (пистолет на место) OR DI\_PortionButton released (if was being held) | (Handled by logic, not state change) Просто pump off while condition holds, resumes if condition clears (no explicit state change, handled within continuous on state). | (remain cont active) |
| Dose mode active (waiting) | CMD\_StartDispDose pressed | (This essentially sets mode active already and likely immediately starts first dose) Start first portion: set internal flag DosePortionRunning=1, turn on pump. Change panel button to "Stop". | Dose portion running |
| Dose portion running (pump ON for dose) | Timer reaches portion time OR DI\_PortionButton released (if ext. button used fully, release stops early) | Turn off pump (DO\_PumpDisp=0); clear DosePortionRunning; increment portion counter; if completed normally, stay in dose mode waiting next portion (with panel showing "Next portion" button active). If ext button was released early or CMD\_StopDispDose pressed, treat as abort -> stop mode entirely (below). | Dose mode paused (waiting next) OR Idle (if aborted) |
| Dose mode paused (waiting next) | CMD\_NextPortion pressed OR external portion button pressed | If pump is off and a next portion command arrives: turn on pump, DosePortionRunning=1 (start next portion). | Dose portion running (pump ON) |
| Dose mode active (any substate) | CMD\_StopDispDose pressed by operator | If a portion currently running: stop pump immediately. If between portions or just ended: simply exit. Mark that dose sequence was terminated early if mid-pump (could log). Set MODE\_DispDoseActive=0. | Idle (Dispense off) |
| Dose mode active (any substate) | Aвария или требование выхода (e.g., user navigates away) | Stop pump if on; MODE\_DispDoseActive=0; if aborted mid-portion, note possibly in log. | Idle (Dispense off) |

**Условия logic continuous (within state)**:

* When in continuous active: DO\_PumpDisp = MODE\_DispContActive AND ( (PAR\_UseGunSensor? !DI\_GunSensor : True) AND (PAR\_UsePortionButton? DI\_PortionButton : True) ).
  + Means pump flows only if cont mode on and gun lifted and ext. button held (if those enabled).
  + If gun placed or button released, pump stops automatically, but state is still cont active – so if they lift gun again or press button again, flow resumes (without needing to press "Pуск" again).
  + That might be intended: gun acts like a trigger on/off while mode active.
* Similarly for dose:
  + If ext. button option: we can incorporate that if portion running: if user releases pedal early, we stop pump (so portion incomplete).
  + If gun sensor: apply similarly to continuous – e.g., if gun placed during portion, maybe stop pump (some spillage risk but at least stops).
  + These are managed by logic rather than state transitions.

**Open questions** on dispensing might be: how exactly ext. button and gun interplay, but we've set a reasonable approach.

Этот раздел (диаграммы состояний) детально описал переходы для всех режимов.

**Каталог переменных (таблицы)**

Ниже приведены таблицы переменных проекта, разделённые на:

* **A) Глобальные переменные** – общесистемные переменные, используемые для обмена между модулями и с внешними системами. Указаны имя, тип, признак Retain, описание (включая владельца/производный модуль и доступ SCADA).
* **B) Локальные переменные модулей** – ключевые внутренние переменные каждого функционального блока (для понимания реализации, они не экспортируются вовне).
* **C) Сетевые переменные** – переменные, сопоставленные Modbus-регистрам (в Slot 1, Modbus Slave интерфейс). Приведены имя, тип, права (R/W) и назначение. *Адреса регистров указаны как TBD*, поскольку согласно требованиям адреса будут определены на этапе конфигурирования; мы сохраняем структуру, но оставляем поля адресов пустыми для заполнения.

**A. Глобальные переменные**

| **Имя переменной** | **Тип** | **Retain** | **Описание и использование (модули, SCADA)** |
| --- | --- | --- | --- |
| **DI\_LevelJacket** | Bool | No | Дискретный вход: датчик уровня воды в рубашке. True = есть вода. Используется в аварийном модуле (ALM\_NoWater) и регуляторе нагрева (блокирует нагрев). (Аппаратный вход, клемма TBD). |
| **DI\_ProductPres** | Bool | No | Дискретный вход: датчик наличия продукта (например, уровень молока). True = продукт присутствует. Используется для блокировки старта пастеризации (если PAR\_UseProductSensor=True) и индикации на панели (футер). (Клемма TBD). |
| **DI\_MixerFault** | Bool | No | Дискретный вход: авария мешалки (сигнал от частотника, X1). True = авария. Обрабатывается Alarm Manager (устанавливает ALM bit4), Actuator Manager отключает DO\_Mixer. (Клемма TBD). |
| **DI\_HeaterFault** | Bool | No | Дискретный вход: авария нагрева (например, термостат на рубашке, X2). True = перегрев/авария. Alarm Manager -> ALM bit5, Actuator Manager снимает DO\_Heater\*. (Клемма TBD). |
| **DI\_GunSensor** | Bool | No | Дискретный вход: датчик положения пистолета розлива. True = пистолет на месте (датчик сработал). Используется в режиме розлива: при True насос отключается. (Клемма TBD). |
| **DI\_PortionButton** | Bool | No | Дискретный вход: внешняя кнопка подачи порции (педаль). True при нажатии. В режиме розлива, при PAR\_UsePortionButton=True: разрешает подачу насоса (для непрерывного) и запускает/продолжает порцию (для дозирования). (Клемма TBD). |
| *(DI\_... остальные)* | Bool | No | **DI\_1...DI\_16** – остальные дискретные входы ПР102. Если не используются в данном проекте, остаются резервом (TBD). Например, можно задействовать для внешнего старт/стоп пастеризации, аварийного стопа и т.д., при необходимости. |
| **AI\_TempProduct** | Int (or Float) | No | Аналоговый вход AI1: датчик температуры продукта (NTC). Конфигурирован под NTC10k; выдает значение, преобразованное в 0,1°С (int) или в °C (float). Программа сохраняет результат в PV\_TempProduct (ниже). Этот AI также формирует ошибочные состояния: при обрыве/КЗ – влияет на ALM bits0/1. |
| **AI\_TempJacket** | Int/Float | No | Аналоговый вход AI2: датчик температуры рубашки (NTC). Аналогично AI1. Значение используется в алгоритме нагрева (для плавного нагрева) и для контроля аварий bits2/3. |
| **AO\_HeaterPower** | Int | No | Аналоговый выход AO1: сигнал на тиристорный регулятор нагрева (0–1000 = 0–100.0%). Управляется регулятором нагрева (PID). **Связана с MB\_HeaterPower (рег.517)** для наблюдения/настройки мощн. тэнов. (Клемма TBD). |
| **AO\_Reserve** | Int | No | Аналоговый выход AO2: не используется (резерв). (Клемма TBD). |
| **DO\_Heater1** | Bool | No | Цифровой транзисторный выход: ТЭН №1. Управляет первой группой нагревателей. При PID-управлении – этот выход может быть постоянно вкл (если тиристор модулем управляется AO), либо пульсирует (при ШИМ). При гистерезисе – просто on/off. Активируется регулятором нагрева. (Клемма TBD, например DO6 согласно бит5). |
| **DO\_Heater2** | Bool | No | Выход: ТЭН №2 (вторая группа нагревателей, контактор). Вкл/выкл регулятором нагрева при высокой потребности. Соответствует DO7 (бит6). |
| **DO\_Heater3** | Bool | No | Выход: ТЭН №3 (третья группа). Вариантно используется, если PAR\_HeaterGroup3Enabled. DO8 (бит7). |
| **DO\_SteamValve** | Bool | No | (Альтернатива нагревателям) Выход: клапан подачи пара. Используется, если PAR\_UseSteam=True вместо DO\_Heater1–3. Соответствует DO1 (бит0), когда сконфигурирован как паровой клапан. В проекте variant1 – электрокалорифер, SteamValve=False. |
| **DO\_PumpCirc** | Bool | No | Выход: циркуляционный насос рубашки. DO1 бит0, если конфигурация = водяное отопление. В нашем случае, рубашка с водой, потому DO1 = насос циркуляции. Активируется при нагреве/охлаждении, чтобы гнать воду по рубашке. В режиме CIP может также использоваться для подачи моющих растворов (см. DO\_PumpCIP). Однако, скорее, CIP имеет отдельный насос. Мы разделим: DO1 – цирк. насос нагрева, DO12 – насос CIP. |
| **DO\_ValveCold** | Bool | No | Выход: клапан охлаждения (подача холодной воды в рубашку). DO2 (бит1). Управляется Temp Manager (Cool\_Demand). Ручной режим тоже управляет. |
| **DO\_MixerFwd** | Bool | No | Выход: мешалка вперёд. DO3 (бит2). Включается при требовании мешалки (авто или ручной). Если реверс не используется, только этот DO будет задействован для мешалки. |
| **DO\_MixerRev** | Bool | No | Выход: мешалка назад (реверс). DO4 (бит3). Используется, если включён опция реверса PAR\_UseMixerReverse. При реверсе, DO\_MixerFwd/Rev включаются взаимоисключающе. В auto-режиме, возможно, не применяется (мешалка только в одну сторону), в ручном – по выбору оператора. |
| **DO\_PumpDispense** | Bool | No | Выход: насос розлива продукта. DO5 (бит4). Управляется модулем Dispense: непрерывно или порционно. |
| **DO\_ValveHot** | Bool | No | (CIP) Выход: клапан ГВС (горячая вода). DO10 (бит9). Открывается CIP Manager на этапах, требующих горячей воды (все этапы ополаскивания/мойки щёлочью/кислотой). |
| **DO\_ValveCold\_CIP** | Bool | No | (CIP) *Использование ХВС во время CIP.* DO9 (бит8) – клапан холодной воды для этапов CIP, когда нужна тёплая вода (смешивание). В коде CIP Manager управляется отдельно от DO\_ValveCold основного охлаждения, хотя физически это тот же клапан (но в нашем проекте DO2 = охлаждение рубашки, а DO9 = клапан ХВС именно CIP линий, возможно). Предположим, рубашка охлаждения и CIP холодная вода – могут быть разные контуры; но, судя по битам, DO9 – на модуле PRM (внешнем), значит отдельный клапан ХВС для CIP. Поэтому введём DO\_ValveCold\_CIP для CIP. |
| **DO\_ValveAlk** | Bool | No | (CIP) Выход: клапан подачи щёлочи. DO11 (бит10). Открывается на этапе щелочной мойки для введения раствора. |
| **DO\_ValveAcid** | Bool | No | (CIP) Выход: клапан подачи кислоты. DO12 (бит11). Открывается на этапе кислотной мойки. |
| **DO\_PumpCIP** | Bool | No | (CIP) Выход: насос циркуляции моющего раствора. DO13 (бит12). Включается на этапах мойки и ополаскивания для прокачки через форсунки/трубопровод. |
| **DO\_ValveDrain** | Bool | No | (CIP) Выход: клапан слива. DO14 (бит13). Открывается для слива отработанной воды/растворов. |
| **MODE\_PasteurActive** | Bool | Yes | Флаг состояния: автоматическая пастеризация активна (режим запущен). Выставляется Mode Manager при старте, сбрасывается по завершению/останову. При True, MB\_SystemStatus bit? (в документации бит0 означает "ручной режим", а для активной пастеризации отдельного бита нет; можно не отображать). Используется для блокировки других режимов. |
| **MODE\_ManualActive** | Bool | Yes | Флаг: ручной режим активирован. Отражает, что пользователь вошёл в режим ручного управления (кнопка на панели). Блокирует автопроцессы. **SCADA:** передаётся как бит0 регистра 518 ("ручной (поэтапный) режим включён"). |
| **MODE\_CIPActive** | Bool | Yes | Флаг: режим мойки (CIP) выполняется. Используется для блокировок (не допустить пастеризацию/розлив одновременно). **SCADA:** нет отдельного бита, но косвенно видно по рег.503 (если ≠0, значит CIP идёт). |
| **MODE\_DispContActive** | Bool | Yes | Флаг: режим непрерывного розлива активен. Устанавливается при нажатии "Пуск розлива", сбрасывается при стопе. Блокирует пастеризацию/мойку. (На SCADA можно не передавать, но при необходимости – можно отразить в MB\_SystemStatus резервным битом). |
| **MODE\_DispDoseActive** | Bool | Yes | Флаг: режим дозированного розлива активен. Аналогично выше, при нажатии "Пуск дозир. розлива". |
| **MODE\_RecipeActive** | Bool | Yes | Флаг: поэтапный (рецептурный) режим активен. Устанавливается при выборе рецепта (опция 7). **SCADA:** возможно, бит0 того же регистра 518 трактуется и как ручной, и как поэтапный (в тексте 518 bit0 подписан "ручной (поэтапный) режим"). То есть, если MODE\_RecipeActive или MODE\_ManualActive – они оба отражаются одним битом. Мы можем использовать один флаг для обоих: либо объединить понятия, либо использовать отдельный, но тогда SCADA его не увидит. Вероятно, поэтапный режим – это и есть "ручной (поэтапный)" в понимании системы (опция1). Поэтому MODE\_ManualActive может означать как чисто ручное, так и поэтапное – путаница. В нашем решении: MODE\_RecipeActive отдельный, а в MB\_SystemStatus bit0 выставляем, если любой из них true (т.е. либо ручной, либо поэтапный). |
| **STEP\_PasteurStage** | Int | No | Текущий этап пастеризации (1–5), или 0 если нет. Устанавливается Temp Manager/Mode Manager. Используется для внутренней логики переходов. Для SCADA используется вместо него битовая маска 504 (формируется отдельно). |
| **STEP\_CIPStage** | Int | No | Текущий этап CIP (1–5 для полной; 1–3 для частичных). Используется CIP Manager. Во внешнюю маску 503 преобразуется. |
| **CMD\_StartPasteur** | Bool | No | Команда запуска пастеризации (автоцикла) – от HMI или внешней системы. Активируется при нажатии кнопки "Пуск" на экране пастеризации. Mode Manager при обнаружении =1 запускает цикл и затем сбрасывает. (SCADA напрямую не пишет эту команду в указанных регистрах, можно реализовать отдельным Coil, но сейчас – только от HMI). |
| **CMD\_StopPasteur** | Bool | No | Команда остановки пастеризации – от HMI (кнопка "Стоп" во время цикла). Mode Manager останавливает процесс по фронту этого сигнала. |
| **CMD\_SkipHeatStage** | Bool | No | Команда пропуска этапа нагрева – активируется, когда время нагрева истекло или равно 0, и оператор нажимает кнопку "Пропустить этап 1". Temp Manager/Mode Manager при =1 сразу переходят к выдержке, устанавливая соответствующие флаги (и ALM13, возможно). |
| **CMD\_StartCIP\_Full** | Bool | No | Команда запуска полной CIP-мойки. От HMI (кнопка "Полная мойка"). CIP Manager/Mode Manager при =1 начинают цикл 5 этапов. |
| **CMD\_StartCIP\_Alk** | Bool | No | Команда запуска щёлочного CIP (3 этапа). |
| **CMD\_StartCIP\_Acid** | Bool | No | Команда запуска кислотного CIP (3 этапа). |
| **CMD\_StartCIP\_Rinse** | Bool | No | Команда запуска ополаскивания (1 этап). |
| **CMD\_StopCIP** | Bool | No | Команда немедленной остановки CIP. От HMI (кнопка "Стоп" на экране мойки). CIP Manager прерывает цикл по этому сигналу. |
| **CMD\_StartDispCont** | Bool | No | Команда запуска непрерывного розлива. От HMI (кнопка "Пуск постоянного розлива"). Mode Manager ставит MODE\_DispContActive=1 на фронте. Может рассматриваться и как состояние (мы в программе используем флаг Mode). Здесь храним команду для четкости; после обработки сбрасываем. |
| **CMD\_StopDispCont** | Bool | No | Команда остановки непрерывного розлива. От HMI (кнопка "Стоп" на экране розлива). Обработка: MODE\_DispContActive=0, останов насоса. |
| **CMD\_StartDispDose** | Bool | No | Команда запуска дозированного розлива. От HMI (кнопка "Пуск дозирующего"). При взведении Mode Manager активирует дозир.режим. Затем автосброс. |
| **CMD\_StopDispDose** | Bool | No | Команда прекращения дозированного режима (кнопка "Стоп" на экране или выход из режима). Приводит к MODE\_DispDoseActive=0, останову насоса если работал. |
| **CMD\_NextDose** | Bool | No | Команда подачи следующей порции (кнопка "Следующая порция"). По фронту Dispense Manager, если насос свободен, запускает новый цикл дозы. Если PAR\_UsePortionButton=True, эта команда может не использоваться (вместо неё педаль). |
| **CMD\_ResetAlarms** | Bool | No | Команда сброса аварий. От панели (кнопка "Сброс"). Alarm Manager при получении =1 очищает сбрасываемые биты в ALM\_Mask (только те, условия которых уже нормализовались), и сбрасывает сам флаг. |
| **PAR\_UseSteam** | Bool | Yes | Параметр конфигурации: использовать паровой клапан вместо электронагревателей (опция14). Если True, вместо управления DO\_Heater1-3, регулятор нагрева управляет DO\_SteamValve (через гистерезис/ШИМ). В variant1 = False (электронагрев). |
| **PAR\_UsePID** | Bool | Yes | Параметр: использовать PID-регулирование нагрева (опция11). True – регулятор работает в аналоговом режиме (AO\_HeaterPower), False – только дискретный режим. В variant1 контроллер поддерживает PID (модификация .2), можно включить = True. |
| **PAR\_HeaterGroup2Enabled** | Bool | Yes | Конфигурация: наличие второй группы ТЭНов (опция12). True – DO\_Heater2 задействован. В variant1 = True. |
| **PAR\_HeaterGroup3Enabled** | Bool | Yes | Конфигурация: третья группа ТЭНов (опция13). True – DO\_Heater3 задействован. В variant1 возможно True (судя по схеме есть К5). Если фактически третий ТЭН отсутствует – поставить False, тогда DO8 не используется. |
| **PAR\_UseMixerReverse** | Bool | Yes | Параметр: использовать реверс мешалки (опция8). Если True, задействован DO\_MixerRev; если False, мешалка всегда в одном направлении (DO\_MixerFwd). variant1 скорее = False (нет реверса, если двигатель асинхронный с частотником, реверс возможен, но необходимость?). Оставим False по умолчанию. |
| **PAR\_UseVFD** | Bool | Yes | Параметр: использование частотника для мешалки (опция9). True – скорость мешалки регулируется (SCADA рег.506 активен), False – мешалка работает с фиксированной скоростью (50 Гц). variant1: имеется частотник, = True. Если False, VFD Manager можно не отправлять команды скорости, а DO\_MixerFwd просто on/off. |
| **PAR\_MaxMixerSpeed** | UInt16 | Yes | Ограничение максимальной скорости мешалки (% от 50 Гц, в десятых). Опция10. Например, 500 = 50,0% (25 Гц). VFD Manager при установке частоты проверяет и ограничивает. По умолчанию 1000 (100%). |
| **PAR\_DispenseEnabled** | Bool | Yes | Параметр: разрешение режима розлива (опция6). False – режим розлива скрыт/недоступен, True – активен. variant1: True. Если False, MODE\_Disp\* никогда не включаются. |
| **PAR\_DosingEnabled** | Bool | Yes | Параметр: включение дозирующего режима (опция 5). False – на панели доступен только непрерывный розлив, дозирование скрыто. variant1: True. |
| **PAR\_UseGunSensor** | Bool | Yes | Параметр: использовать датчик пистолета (опция4). True – логика автопуска/остановки насоса по датчику активна. variant1: предположительно True (если датчик предусмотрен). |
| **PAR\_UsePortionButton** | Bool | Yes | Параметр: использовать внешнюю кнопку порции (опция5). True – насос в режимах розлива работает только при нажатии внешней кнопки. variant1: предположительно True (если педаль). |
| **PAR\_PID\_Kp** | Float | Yes | ПИД: коэффициент пропорциональный. Настраивается в меню "Автонастройка ПИД" или вручную. Используется регулятором нагрева. |
| **PAR\_PID\_Ti** | Float | Yes | ПИД: постоянная интегрирования (сек). (или Ki). |
| **PAR\_PID\_Td** | Float | Yes | ПИД: постоянная дифференцирования. |
| **PAR\_PumpFlowRate** | Float | Yes | Параметр: производительность насоса розлива (мл/с). Определяется калибровкой (в "Настройки розлива" оператор измеряет, используя непрерывный режим). Используется для расчета времени дозы. По умолчанию, например, 100 мл/с (зависит от оборудования). |
| **SP\_HeatTemp** | Int16 | Yes | Уставка температуры пастеризации, 0,1 °С. (Регистр 510). Определяет целевую температуру нагрева продукта. По умолчанию, например, 680 (68,0°C). Доступна для изменения на HMI/SCADA (R/W). |
| **SP\_HeatTime** | UInt16 | Yes | Время нагрева (минуты). (Регистр 511). Если >0, задает лимит времени на этап нагрева. 0 = без ограничения (на экране "Время нагрева: 0 мин" интерпретируется как нет таймаута, разрешая пропуск этапа вручную). По умолчанию 0 (нагрев до достижения Т, даже если долго). |
| **SP\_SmoothTemp** | Int16 | Yes | Порог плавного нагрева продукта, 0,1°C. (Регистр 512). Когда температура продукта превышает этот уровень, начинает действовать режим "плавного" нагрева: снижается температура рубашки до СП нагрева продукта (либо ограничивается разность). По умолчанию 500 (50,0°C). |
| **SP\_SmoothDiff** | Int16 | Yes | Дифференциал рубашки при плавном нагреве, 0,1°C. (Регистр 513). Определяет максимально допустимую разницу (рубашка выше продукта) после перехода в плавный режим. Например, 20 (2,0°C) по умолчанию. В нашей логике: до SP\_SmoothTemp рубашка могла быть нагрета выше, после – поддерживаем ≈ SP\_HeatTemp + SmoothDiff. |
| **SP\_HoldTime** | UInt16 | Yes | Время выдержки продукта, с. (Регистр 514). Длительность этапа 2 пастеризации. По умолчанию 600 с (10 мин) для пастеризации 68°С. |
| **SP\_HoldDiff** | Int16 | Yes | Дифференциал нагрева при выдержке, 0,1°C. (Регистр 515). Допустимое падение температуры продукта перед включением нагрева на этапе выдержки. Например, 10 (1,0°C). Наша логика: если продукт остыл на 1,0°C ниже уставки, разрешаем рубашке нагреться на ~1°C выше продукта. |
| **SP\_CoolTemp** | Int16 | Yes | Температура охлаждения продукта, 0,1°C. (Регистр 516). Целевая температура готового продукта после охлаждения. По умолчанию, например, 270 (27,0°C). Используется Temp Manager для этапа охлаждения. |
| **SP\_HeaterPower** | Int16 | Yes | "Мощность нагревательных элементов", 0,01%. (Регистр 517). Этот параметр – пользователь вводит процент от полной мощности тэнов (в примере 10,12%). Предположительно 10000 = 100,00%. Используется справочно (возможно, для расчета нагрева или отображения). В нашем случае сохраняем и отображаем на SCADA. Регулятор напрямую его не использует, либо мог бы ограничивать PID-выход в процентах (например, чтоб не превышать 80% если питание проседает, но не указано). |
| **SP\_CIP\_RinseTime** | UInt16 | Yes | Настройка: время этапа ополаскивания CIP, сек. (для полного CIP этап1,3,5; для укороченных – этап1 и финал). Например, 180 с. Можно иметь раздельно: SP\_CIP\_RinseTime1, SP\_CIP\_RinseTimeBetween, SP\_CIP\_RinseTimeFinal, но для упрощения – одно значение, если нужно точнее – можно разделить. |
| **SP\_CIP\_AlkTime** | UInt16 | Yes | Время циркуляции щёлочи, сек. (этап2 полного/этап2 щелочного CIP). Например, 600 с. |
| **SP\_CIP\_AcidTime** | UInt16 | Yes | Время циркуляции кислоты, сек. (этап4 полного/этап2 кислотного CIP). Например, 600 с. |
| **SP\_DoseVolume** | UInt16 | Yes | Объём порции продукта для дозированного розлива. Единицы – мл (или условные). Оператор задаёт на экране (п.4). PLC использует для вычисления времени работы насоса: DoseTime = SP\_DoseVolume / PAR\_PumpFlowRate. Например, 500 мл. (Modbus-адрес не предусмотрен, HMI пишет напрямую). |
| **MB\_AlarmMask** | UInt16 | No | Маска аварий (Modbus-регистр 500). Формируется Alarm Manager из бит ALM\_\*. Каждый бит соответствует состояниям, перечисленным ранее. SCADA читает этот регистр (R). Сброс аварий идёт через CMD\_ResetAlarms. |
| **MB\_TempProduct** | Int16 | No | Температура продукта, 0,1°C (Modbus 501). Обновляется Temp Manager из PV\_TempProduct. SCADA/HMI читают (R). Поддерживает отрицательные (NTC8016: -1,5°C -> 65521). |
| **MB\_TempJacket** | Int16 | No | Температура рубашки, 0,1°C (Modbus 502). R. |
| **MB\_CIP\_Stages** | UInt16 | No | Маска этапов мойки (Modbus 503). Каждый бит 0-11 указывает, какой цикл и этап CIP выполняется. Формируется CIP Manager: при старте цикла устанавливается соответствующая группа бит. Например, при полной мойке: этап1 -> bit0=1, затем бит0->0, bit1=1, и т.д. При щелочной: bit5-bit7, кислотной: bit8-10, ополаск: bit11. SCADA может читать для отображения прогресса CIP. |
| **MB\_PasteurStages** | UInt16 | No | Маска этапов пастеризации (Modbus 504). Биты: 0=нагрев,1=выдержка,2=охлаждение,3=выравнивание,4=хранение,5=завершено. Устанавливается Temp/Mode Manager: в каждый момент активен один из бит0-4; по завершении цикла – бит5=1. SCADA читает для индикации на мнемосхеме. |
| **MB\_OutStateMask** | UInt16 | No | Маска состояний выходов (Modbus 505). Формируется Actuator Manager на основе текущих DO: бит0=DO1, ..., бит13=DO14. SCADA читает для контроля. Замечание: Bit0 означение зависит от конфигурации: "насос циркуляции или клапан пара" – наш Actuator Manager ставит этот бит, когда либо PumpCirc либо SteamValve активен (в зависимости от PAR\_UseSteam). |
| **MB\_SetMixerFreq** | UInt16 | R/W | Заданная частота мешалки, в 0,1% от 50 Гц (Modbus 506). Диапазон 0–1000 (значение 500 = 25 Гц). Запись: SCADA/HMI могут устанавливать скорость (%). Чтение: SCADA может контролировать текущее задание. VFD Manager периодически отсылает это значение ПЧ. Retain для этого не нужен (стоит по умолч. 1000, можно ограничено PAR\_MaxMixerSpeed). |
| **MB\_TimeDisplayMode** | UInt16 | W | Режим отображения времени мойки (Modbus 507). Код, задаваемый HMI: 0 = "До конца мойки", 1 = "До конца этапа", 2 = "Прошло времени". CIP Manager читает значение при обновлении MB\_WashTimeMin/Sec. SCADA может при желании тоже установить (но вряд ли нужно). |
| **MB\_WashTimeMin** | UInt16 | R | Время мойки - минуты (Modbus 508). Отображает либо остаток/прошедшее время (в мин) в соответствии с режимом 507. Обновляется CIP Manager каждую секунду. SCADA читает (R). |
| **MB\_WashTimeSec** | UInt16 | R | Время мойки - секунды (Modbus 509). Младшие 0–59 с соответствующего таймера. |
| **MB\_HeatTemp\_SP** | Int16 | R/W | Уставка температуры нагрева продукта (Modbus 510). Привязана к SP\_HeatTemp. Retain. SCADA R/W. |
| **MB\_HeatTime\_SP** | UInt16 | R/W | Время нагрева (мин) (511). Привязка к SP\_HeatTime. SCADA R/W. |
| **MB\_SmoothTemp\_SP** | Int16 | R/W | Порог плавного нагрева (512) -> SP\_SmoothTemp. |
| **MB\_SmoothDiff\_SP** | Int16 | R/W | Дифференциал рубашки (плавный) (513) -> SP\_SmoothDiff. |
| **MB\_HoldTime\_SP** | UInt16 | R/W | Время выдержки, с (514) -> SP\_HoldTime. |
| **MB\_HoldDiff\_SP** | Int16 | R/W | Дифференциал на выдержке (515) -> SP\_HoldDiff. |
| **MB\_CoolTemp\_SP** | Int16 | R/W | Температура охлаждения (516) -> SP\_CoolTemp. |
| **MB\_HeaterPower** | Int16 | R/W | Мощность нагревателей (517) -> SP\_HeaterPower. Шкала 0,01%. SCADA R/W. |
| **MB\_SystemStatus** | UInt16 | R | Статус системы (битовая маска, 518). Пока определён только bit0 - "ручной (поэтапный) режим включён". Мы выставляем бит0 =1, если MODE\_ManualActive или MODE\_RecipeActive true. Остальные биты не используются (0). SCADA читает для общей индикации. |

*(Примечание: имена MB\_... выбраны для ясности привязки к Modbus, фактически можно обходиться теми же PV\_, SP\_ переменными. Мы декомпозировали некоторые – напр. MB\_OutStateMask фактически формируется из DO\_..., можно хранить только локально. Но для удобства SCADA/CSV делаем явную переменную.)*

**B. Локальные переменные по модулям**

**Менеджер режимов:**

* currentMode (Int) – код активного режима (0=Idle, 1=Pasteur, 2=Manual, 3=CIP, 4=DispCont, 5=DispDose, 6=Recipe). Используется для логики переключения, альтернатива булевым ModeActive флагам. Мы храним флаги Retain, но runtime можно использовать эту переменную для единственности.
* pendingMode (Int) – буфер для запроса режима, если пришла команда на запуск режима, пока другой не завершён. Например, если CMD\_StartCIP поступил во время пастеризации, Mode Manager может сохранить pendingMode=3 и выполнить после окончания текущего. (По желанию).
* stopRequest (Bool) – флаг запроса остановки, общий. Например, Emergency Stop. Если true, Mode Manager останавливает любой режим.
* manualEntered (Bool) – флаг, что пользователь вошёл в ручной режим (например, переход с главного меню). Это аналог MODE\_ManualActive, но Mode Manager мог бы дополнительно отличать, поэтапный или полный ручной.
* recipeSelected (Int) – ID выбранного рецепта (если реализуем). For multiple recipe steps mode.

**Температурный менеджер:**

* PV\_TempProduct (Float) – текущая температура продукта, °C (или Int deci-deg). Читается из AI\_TempProduct (с калибровкой, если нужно). Используется в регуляции и для MB\_TempProduct (после конвертации).
* PV\_TempJacket (Float) – текущая температура рубашки, °C.
* Heat\_Demand (Float 0-100%) – требуемая мощность нагрева, расчет Temp Manager. В дискретном случае =100% or 0%. В PID – можно передавать error magnitude. Но мы скорее напрямую PID используем, так что Heat\_Demand – промежуточный, не нужен, если PID fbk is integrated.
* Cool\_Demand (Bool) – требование включить охлаждение. True, если надо охлаждать (product > SP\_CoolTemp).
* StageTimer (Timer / int) – таймер текущего этапа (нагрев, выдержка, etc). Отсчитывает время до конца этапа (HoldTime etc).
* heatTimeout (Bool) – флаг истечения времени нагрева. Используется для отображения кнопки пропуска.
* holdTimeRemaining (UInt) – оставшееся время выдержки (для внутреннего использования; SCADA не просит, но можно отобразить на панели).
* storeActive (Bool) – флаг, что режим хранения выполняется (Stage5). = PAR\_StoreEnabled копия при переходе.
* ProductTempMax (Float) – буфер: целевая температура пастеризации. Может обновляться, если SP changed during process (в принципе, SP\_HeatTemp).
* JacketTargetTemp (Float) – желаемая температура рубашки в данный момент. Например,
  + На нагреве: initially, JacketTarget = (overshoot allowed?) e.g. SP\_HeatTemp + X (if we allowed overshoot).
  + After SmoothTemp: JacketTarget = SP\_HeatTemp + SP\_SmoothDiff.
  + On hold: JacketTarget maybe = PV\_Product + SP\_HoldDiff.
  + Temp Manager uses this to tell Regulator/PID the setpoint for jacket (if controlling analog SSR with separate loop) – but we have one PID for product, so maybe not used.
* smoothMode (Bool) – флаг, что система перешла в плавный нагрев. Для логирования / one-time actions (like reduce SSR threshold).
* levelingTimer (Timer) – таймер выравнивания, e.g., fixed short time.

**Регулятор нагрева:**

* PID\_Output (Float) – текущее расчетное управляющее воздействие (0–100%).
* PID\_Error (Float) – текущая ошибка (SP\_HeatTemp - PV\_TempProduct).
* Integrator, Differentiator etc – внутренние sum and prev error for manual PID code, if we implement.
* HysteresisState (Bool) – состояние выхода гистерезисного регулятора (true = on). Determined by threshold logic.
* CycleTimer (Timer) – таймер цикла ШИМ, если применяем PWM control for contactor to approximate lower power. E.g., 10s cycle, turned on fraction of time.
* AllHeatersOff (Bool) – flag all heater outputs off. If something triggers to off.
* Alarm\_PID\_Coeff (Bool) – flag if PID coefficients invalid (used to raise ALM bit7).
* useAnalog (Bool) – local copy of PAR\_UsePID.
* steamMode (Bool) – local copy of PAR\_UseSteam.
* groups (Int) – number of heater groups enabled (1,2,3) from PAR\_HeaterGroup2/3.
* (He might also have info about SSR linearization or similar, but not needed.)

**Менеджер исполнительных механизмов:**

* DO\_State[1..14] (Bool array) – текущее состояние каждого физического DO (maybe not needed if directly controlling outputs via variables, but could maintain an image). For building MB\_OutStateMask easily.
* Interlocks (various bools) – e.g., lockHeating (true if cooling active or water low), lockMixer (if fault).
* pendingReverse (Bool) – internal flag when switching mixer direction: set when one direction turned off, short delay then enable other.
* reverseDelayTimer (Timer) – for above.
* coolingOverride (Bool) – if cooling forced off heating.
* ManualHeat1\_Cmd, ManualHeat2\_Cmd, ManualCool\_Cmd – copies of manual commands, used to implement exclusivity:
  + For example, if ManualCool\_Cmd goes true, Actuator sets ManualHeat1\_Cmd and ManualHeat2\_Cmd to false (and thus DO off).
* Could mirror CMD\_ManMixerOn, CMD\_ManHeat1On,... etc, but those are global (we left them out to minimize global count; actually in global we did not define CMD\_ManHeat1On etc, we assumed manual toggles happen directly to DO by panel? Possibly the panel toggles global flags as well).
  + Actually, in OwenLogic, panel might directly manipulate DO variables if exposed, but better to have variables:
    - We didn't list CMD\_ManMixerOn, CMD\_ManHeat1On, etc as global. Perhaps should:
      * Because HMI likely writes to some variable for those toggles.
      * But we can consider that HMI, in manual control screen, is bound to the DO variable directly (like pressing button sets DO output through network variable or internal).
      * But since we want logic (interlock) in between, we should have intermediate variables.
      * So ideally define:  
        CMD\_Manual\_MixerOn, CMD\_Manual\_Heater1On, CMD\_Manual\_Heater2On, CMD\_Manual\_CoolingOn.
      * For brevity earlier, we omitted them from global. We can treat them as local in Actuator manager, updated from panel as network or internal, but easier to manage if global:  
        Possibly panel sets global (like how panel uses modbus with internal variables).
      * Given time, let's assume Actuator manager reads some system variables for those (like those DO might be mapped to panel toggles, which we intercept).
      * We'll include them conceptually:
        + Not in table A to not clutter more, but as needed local references or global if required. It's an implementation detail if Owen requires mapping manual toggles to variables or can the DO be tied directly to HMI element with script logic for conditions? It's likely simpler to have variables.
* blinkIndicator (Bool) – maybe used for blinking level indicator if no water (footage mentions blinking red-blue if water absent – but likely panel does that seeing ALM\_NoWater).
* OutMask (UInt16) – assembled output mask.

**Modbus Master (VFD Manager):**

* ModbusPollTimer (Timer) – to schedule periodic polls to VFD (if not using internal polling mechanism).
* vfdResponseOK (Bool) – flag to indicate last modbus read success. If fails multiple times, raise comm error.
* noResponseCount (Int) – counter of sequential comm failures. If >N, raise ALM\_MixerFault.
* vfdStatusWord (UInt16) – last read status from VFD (could decode bits like fault, running, etc).
* vfdFaultActive (Bool) – from status or separate indicator coil, indicates VFD fault condition (set ALM\_MixerFault).
* vfdFreqFeedback (UInt16) – actual frequency from VFD if read (we didn't plan to output it anywhere, but could if needed).
* SetFreqLastSent (UInt16) – track last frequency sent to avoid redundant writes.

**SCADA/Slave Manager:**

* Not much internal needed; maybe just uses built-in mapping by name.

**Alarm Manager:**

* ALM\_SensorProd\_Break (Bool) – input from analog module or manual check (AI out-of-range).
* ALM\_SensorProd\_Short (Bool).
* ALM\_SensorJacket\_Break, ALM\_SensorJacket\_Short.
* ALM\_MixerFault, ALM\_HeaterFault, ALM\_PowerFail, ALM\_PIDFault, ALM\_NoWater, ALM\_PasteurAbort, ALM\_CIPAbort, ALM\_CommExpFault.
* Actually we had them implicitly via mask bits in table21; define each as booleans for internal logic, combined to mask word.
* prevAlmMask (UInt16) – to detect rising edges for log entries.
* retainWasAuto (Bool) – saved Retain flag set to true when an auto-cycle starts, to detect power fail (if PLC reboots and finds it true, it sets ALM\_PowerFail).
* AlarmLogIndex, EventLogIndex, DoseLogIndex (Int) – indices for log arrays (circular).
* AlarmLog[50] (structure with code + timestamp) – store records.
* EventLog[...].
* DoseLog[...].
* Possibly arrays as global to allow SCADA reading if needed, but user didn't specifically ask, so mention conceptual only.

**Settings Manager:**

* FirstStart (Bool, Retain) – marker that defaults loaded (we mention STAT\_FirstStart earlier).
* Possibly not needed if we rely on default retentive values in CSV.
* But often you'd see if FirstStart != magic value, then set defaults and write FirstStart= True, to not override on next boot.
* In OwenLogic, Retain might persist once set by CSV default, so maybe done differently. But be safe:
* initDone (Bool) – internal to ensure one-time initialization done in code.

**Timer/RTC Manager:**

* autoStartTime (Time struct or two bytes for hours/min) – from user setting (maybe in service menu).
* autoStartEnabled (Bool).
* kitchenTimerSet (Time struct HH:MM:SS).
* kitchenTimerRun (Bool).
* kitchenTimerRemaining (Time).
* kitchenTimerDone (Bool).
* Perhaps kitchenTimerActive (blinking indicator).
* Or simpler:
  + Use one of PLC's internal RTC function to countdown given time; or implement downcounter in code.
* RTC\_Now (DateTime) – read current time.

*(Для ясности, выше – локальные, потому без Retain, кроме retainWasAuto etc as needed).*

**C. Сетевые переменные (Modbus Slave, Slot 1)**

Ниже приведён список переменных, привязанных к Modbus-регистрам (адреса TBD). Все указаны как Network переменные OwenLogic. Комментарии отражают их назначение, масштаб и права доступа. Адреса и метки соответствуют описанной карте регистров SCADA.

| **Имя переменной** | **Тип** | **Адрес регистра (TBD)** | **R/W** | **Описание (Modbus-тег и единицы)** | **Метатип** |
| --- | --- | --- | --- | --- | --- |
| MB\_AlarmMask | UInt16 | 500 (TBD) | R | Маска аварий: бит0 обрыв прод.датч., 1 КЗ прод., 2 обрыв руб., 3 КЗ руб., 4 мешалка авария, 5 нагрев авария, 6 питание потеряно, 7 PID ошибка, 12 нет воды, 13 стоп пастер., 14 стоп мойки. | Network |
| MB\_TempProduct | Int16 (signed) | 501 (TBD) | R | Температура продукта, 0.1 °C (значение 255 = 25.5°C). Отрицательные представлены как unsigned (65521 = -1.5°C). | Network |
| MB\_TempJacket | Int16 | 502 (TBD) | R | Температура рубашки, 0.1 °C (значение 42 = 4.2°C, 65521 = -1.5°C). | Network |
| MB\_CIP\_Stages | UInt16 | 503 (TBD) | R | Этапы мойки (битовая маска). Биты: 0–4 полная мойка этап1–5; 5–7 щёлочная этап1–3; 8–10 кислотная этап1–3; 11 ополаскивание этап1; остальные рез. 1 = соответствующий этап выполняется. | Network |
| MB\_PasteurStages | UInt16 | 504 (TBD) | R | Этапы пастеризации (маска). bit0=нагрев,1=выдержка,2=охлажд,3=выравн.,4=хранение,5=завершено. В каждый момент один из bit0-4 =1; по завершении цикла bit5=1. | Network |
| MB\_OutStateMask | UInt16 | 505 (TBD) | R | Состояние выходов (маска). Bit0 = DO1 (насос цирк. или паров. клапан), 1 = DO2 (клапан охлажд.), 2 = DO3 (мешалка вперёд), 3 = DO4 (меш.рев), 4 = DO5 (насос розлива), 5 = DO6 (ТЭН1), 6 = DO7 (ТЭН2), 7 = DO8 (ТЭН3), 8 = DO9 (клапан ХВС CIP), 9 = DO10 (клапан ГВС), 10 = DO11 (щелочь), 11 = DO12 (кислота), 12 = DO13 (насос CIP), 13 = DO14 (слив). 1 = выход включён. | Network |
| MB\_SetMixerFreq | UInt16 | 506 (TBD) | R/W | Заданная частота мешалки, в 0.1% от 50Hz. Диапазон 0...5000 (предположительно поддержка до 500%, но обычно 0–1000). Пример: 500 = 50.0% = 25Hz. SCADA может устанавливать требуемую скорость (HMI тоже пишет при изменении). | Network |
| MB\_TimeDisplayMode | UInt16 | 507 (TBD) | W | Режим отображения времени мойки. 0 = "До конца мойки", 1 = "До конца этапа", 2 = "Прошло времени". Записывается HMI в зависимости от выбора пользователя. Используется для расчёта MB\_WashTimeMin/Sec. | Network |
| MB\_WashTimeMin | UInt16 | 508 (TBD) | R | Время мойки – минуты. Отображает либо оставшееся, либо прошедшее время (в мин), согласно MB\_TimeDisplayMode. CIP Manager обновляет каждую секунду. | Network |
| MB\_WashTimeSec | UInt16 | 509 (TBD) | R | Время мойки – секунды (0–59). Используется совместно с MB\_WashTimeMin для отображения таймера мойки. | Network |
| MB\_HeatTemp\_SP | UInt16 | 510 (TBD) | R/W | **Уставка температуры нагрева** продукта. Значение = температура (°C) \* 10. Пример: 455 = 45.5°C. Связан с SP\_HeatTemp. Retain, изменяемо SCADA/HMI. | Network |
| MB\_HeatTime\_SP | UInt16 | 511 (TBD) | R/W | **Время нагрева**, мин. Сколько минут отводится на разогрев. 0 = не задано (пропуск этапа при запуске). Связан с SP\_HeatTime. |  |
| MB\_SmoothTemp\_SP | UInt16 | 512 (TBD) | R/W | **Плавный нагрев после** – температура продукта, после превышения которой нагрев рубашки снижается до минимального дифференциала. В 0.1°C. Связан с SP\_SmoothTemp. |  |
| MB\_SmoothDiff\_SP | UInt16 | 513 (TBD) | R/W | **Дифференциал рубашки при плавн.нагреве**, 0.1°C. Максимальная разница температур рубашки над продуктом в плавном режиме. Связан с SP\_SmoothDiff. |  |
| MB\_HoldTime\_SP | UInt16 | 514 (TBD) | R/W | **Время выдержки**, сек. Длительность удержания температуры. Связано с SP\_HoldTime. |  |
| MB\_HoldDiff\_SP | UInt16 | 515 (TBD) | R/W | **Дифференциал при выдержке**, 0.1°C. Допустимый спад температуры продукта до включения нагрева на этапе выдержки. SP\_HoldDiff. |  |
| MB\_CoolTemp\_SP | UInt16 | 516 (TBD) | R/W | **Температура охлаждения** продукта, 0.1°C. До какой температуры охладить после выдержки. SP\_CoolTemp. |  |
| MB\_HeaterPower | UInt16 | 517 (TBD) | R/W | **Мощность нагревателей**, 0.01%. 10000 = 100.00%. Пример: 1012 = 10.12%. Пользовательским параметром характеризует мощность тэнов. Связан с SP\_HeaterPower. |  |
| MB\_SystemStatus | UInt16 | 518 (TBD) | R | **Статус системы** (разное). bit0 = Ручной/поэтапный режим включён. Остальные биты резерв (0). Мы устанавливаем бит0=1, если MODE\_ManualActive or MODE\_RecipeActive. SCADA может мониторить (например, чтобы знать, что система не в авто). |  |

*(Адреса TBD: при импорте CSV эти переменные будут помечены как сетевые, адреса колонка останется пустой или "TBD" для дальнейшего заполнения разработчиком в OwenLogic UI.)*

**CSV файлы**

Ниже представлены содержимые двух CSV-файлов, подготовленных для импорта в среду OwenLogic:

1. **Стандартные\_Переменные.csv** – перечень всех переменных проекта с их основными свойствами.
2. **Сетевые, Слот 1.csv** – список сетевых переменных (Modbus), соответствующих регистровой карте.

**Важно:** столбцы адресов, номеров регистров и клемм – оставлены пустыми (TBD) согласно требованиям. После импортирования разработчик должен назначить точные адреса Modbus и привязать физические входы/выходы к клеммам контроллера.

**Файл: "Стандартные\_Переменные.csv"**

*(Содержимое файла Standard Variables CSV – переменные с типами, Retain и комментариями)*

*Примечание: В CSV перечислены все глобальные переменные из каталога выше. Столбцы: Имя; Тип; Энергонезависимость; Значение по умолчанию; Использование в проекте; Комментарий; Метатип; Путь (Ram/Network). Путь для всех стандартных – "Ram".*

**Файл: "Сетевые, Слот 1.csv"**

*(Содержимое файла Network Slot1 CSV – сетевые переменные с адресами TBD)*

*Примечание: Столбцы: Имя; Тип; Адрес регистра; Комментарий; Метатип; Путь. Адреса регистров оставлены пустыми (TBD) в соответствии с заданием. Метатип всех – "Network".*

**Список открытых вопросов и допущений**

В ходе разработки архитектуры были выявлены некоторые нюансы, требующие подтверждения или уточнений:

1. **Алгоритм смешивания горячей/холодной воды в CIP:** В графиках мойки теплота воды достигается одновременной работой клапанов ХВС и ГВС, однако точная логика (пропорции, время перекрытия) не описана. Мы допустили упрощённую последовательность (сначала обе открыты коротко, потом одна) для получения тёплой воды. При реализации потребуется уточнить, как контролировать температуру моющей воды – возможно, по датчику или просто по времени.
2. **Режим циркуляции моющих растворов:** Предполагается, что мойка выполнена без рециркуляции растворов (химия одноразовая, сразу сливается). В итоге клапан слива DO14 открыт почти на всех этапах щёлочи/кислоты. Это упростило логику (не нужно хранить и повторно использовать растворы). Если же установка подразумевает замкнутый цикл (когда раствор собирается в емкость и используется несколько минут), то алгоритм CIP надо скорректировать: закрывать слив на время циркуляции, затем открывать. Наше решение отражает подход "подал – сразу слил". Требуется подтверждение от технологов.
3. **Датчик продукта (опция 16):** В карте Modbus битов аварий отсутствует сигнал "нет продукта" (что логично – это не авария). Мы предусмотрели DI\_ProductPres и его использование: например, не начинать пастеризацию, если продукта нет, и скрывать/показывать индикатор на панели. Нужны уточнения, требуется ли аппаратный датчик продукта и какова его роль: блокировать нагрев или просто информировать оператора. Мы реализовали блокировку старта цикла при отсутствии продукта (когда PAR\_UseProductSensor=True).
4. **Третий нагреватель (опция 13):** В интерфейсе ручного режима описаны только "Нагрев №1" и "Нагрев №2". Вероятно, в конфигурации Вариант 1 третья группа ТЭНов не предусмотрена, либо "Нагрев №2" сразу включает вторую и третью группу вместе. Мы заложили параметр PAR\_HeaterGroup3Enabled – если False, система будет оперировать только двумя группами (что соответствует UI). Требуется подтвердить фактическое число групп нагрева на данном оборудовании.
5. **Управление частотником мешалки:** Согласно исходным данным, частота задаётся регистром 506, а сигналы "мешалка вперёд/реверс" – через DO3/DO4. Это означает, что старт/стоп мешалки идёт дискретно, а скорость – по Modbus. Мы так и сделали (гибридное управление). Возможно, частотник мог бы запускаться/останавливаться тоже по Modbus, но в проекте оставлены DO для этого (надежнее для аварийных ситуаций). Предполагаем, что такой способ приемлем. Альтернативно, DO3 мог вообще не использоваться, если запуск по Modbus – но тогда при потере связи сложно гарантировать останов. Уточнение: нужно ли помимо задания скорости по сети также по сети давать команду "Run"? (Частотники ОВЕН, напр., могут в режим управления "по скорости" требовать завести дискрету "разрешение хода"). Мы предпочли физический сигнал пуск с DO3.
6. **Раздельность насосов циркуляции рубашки и CIP:** В схеме выходы PRM DO5 и DO1 оба подписаны "циркуляционный насос", что может значить, что один насос используется и для рубашки, и для CIP. В нашем дизайне мы считали DO\_PumpCirc для нагрева/охлаждения рубашки (PR200 DO1), а DO\_PumpCIP – отдельный (PRM DO5). Если фактически насос один, то PRM DO5 может не существовать, а PR200 DO1 будет работать и на CIP, переключаясь на соответствующие линии. Это усложнит логику (одним насосом две функции). Мы оставили их раздельно, предполагая, что CIP-мойка требует дополнительного насоса (например, для разбрызгивающей головки). Необходимо уточнить компоновку: если насос тот же, следует использовать DO1 в CIP Manager вместо DO13, и скорректировать маску битов (bit12 PRM DO5 был бы неиспользован).
7. **Внешняя кнопка порции vs. кнопка "Следующая порция":** Мы интегрировали педаль так, что при её наличии оператор может, удерживая её, выполнить непрерывный розлив, а для дозированного – каждый короткий нажим запускает одну порцию. Однако, текст [14†L95-L100] подразумевает, что при наличии педали *и непрерывный, и дозирующий будут только при нажатии*. Возможно, для дозирующего это означает: даже первая порция не начнётся, пока не нажата кнопка (т.е. "Пуск дозы" на панели готовит систему, а физическое нажатие – выдаёт порцию). Мы это учли частично: насос включается по условию DI\_PortionButton. Но расхождение с "Следующая порция" – unclear: используется ли экранная кнопка 7 вообще, если педаль есть? Предполагаем, что нет – оператор может просто жать педаль каждый раз. В нашем решении, если опция внешней кнопки активна, экранная "Следующая порция" не обязательна (можно её игнорировать или панель скрывает). Это стоит подтвердить, чтобы однозначно реализовать.
8. **Пропуск этапа нагрева (таймаут):** Руководство говорит: "*эта кнопка появляется, если время нагрева истекло или отсутствует*.\*" Мы сделали: если SP\_HeatTime=0 (отсутствует), кнопка "Пропустить нагрев" будет сразу доступна, позволяя начать выдержку мгновенно (для случаев, когда продукт уже горячий или хотят сразу стартовать с выдержки). Если SP\_HeatTime > 0, кнопка появится только когда время вышло, но продукт не достиг SP. Требуется подтверждение, что система действительно позволяет пропустить недогретый продукт к выдержке – это технологически сомнительно (пастеризация не выполнена), но, возможно, на случай, если решили пастеризовать при другой температуре. Вероятно, этот функционал предусмотрен для старта выдержки немедленно, если SP\_HeatTime=0 (т.е. продукт начальной температуры уже >= требуемой, нагрев не нужен). Наш алгоритм оставляет решение оператору. Это поведение лучше согласовать с технологом.
9. **Логирование и журналы:** Предусмотрены журналы аварий, событий, доз – однако их хранение и доступ не оговорены в Modbus-карте. Скорее всего, эти журналы реализованы на панели оператора (Weintek имеет свой лог). Мы описали концепцию хранения в PLC, но без экспонирования по Modbus (что могло бы потребовать десятки регистров). Если SCADA тоже требует журналы, нужно будет спроектировать передачу (например, выдавать последний N событий по запросу). Пока эту задачу мы выносим за скобки – предполагаем, что SCADA достаточно масок и основных параметров, а подробные журналы можно смотреть на HMI.
10. **Совмещение ручного и поэтапного режима:** В статус-регистре 518 бит0 назван "ручной (поэтапный) режим". Вероятно, поэтапный режим – подвид ручного. У нас MODE\_ManualActive и MODE\_RecipeActive разделены. Возможно, их не надо разделять: *опция1 ручной режим нагрева и охлаждения* – это просто ручной; *опция7 автоматический режим по рецепту* – это совсем отдельное (скорее, дополнительный автоцикл). Но 518 бит0 может относиться только к первому. Если поэтапный режим (например, варка сыра) был бы активен, нужно ли ставить бит0? Не ясно. Мы решили устанавливать бит0 при любом "неавтоматическом" управлении (будь то чисто ручной или шаговый). Это стоит уточнить. Если режим рецептов предполагает совсем другую логику (несколько этапов, возможно использующие пастеризацию как под-этап), его реализация выходит за рамки нашей схемы – мы заложили лишь флаг и возможность.
11. **Масштаб регистра 506:** В документе дважды упоминается "коэффициент отображения 0,1, отображается в % от 50 Гц" и пример 25Hz->500. Мы трактовали 500 = 50%, 1000 = 100% (50Hz). Возникает вопрос: почему диапазон регистров 0–5000 указан в задании? Возможно, это опечатка или задел на 5x overspeed. Мы использовали 0–1000 как практический диапазон (0–100%). Требуется подтвердить, не ожидается ли реально 5000 = 250Hz (что маловероятно для мешалки).
12. **Реакция на пропадание связи с HMI/SCADA:** Специальный бит в маске аварий для панели не предусмотрен (bit15 в маске отнесён к расширению ПРМ). Если панель Weintek выступает Master, PLC-Slave не узнает, что связь потеряна. Обычно панель сама оповещает оператора. Мы оставили bit15 резервным, без перепрофилирования. Если важно логировать потерю связи SCADA, можно было бы использовать bit15, но по ТЗ это не требуется, поэтому bit15 остаётся неиспользован.
13. **Привязка входов/выходов к клеммам:** В CSV адреса клемм отмечены "TBD". При внедрении следует свериться с электрической схемой (например, X3 – датчик уровня, X2 – авария ТЭНа, X1 – авария мешалки). Мы сделали разумные соответствия: DI\_LevelJacket -> X3, DI\_HeaterFault -> X2, DI\_MixerFault -> X1, что согласуется с надписями схемы. Просьба подтвердить правильность распиновки.
14. **Дефолты параметров:** Мы заложили примерные значения (например, 68.0°C, 10 мин выдержка, 27.0°C охлаждение) согласно руководству. Если требуются другие по умолчанию, их можно поправить. Retain обеспечивает сохранность последнего установленного, но стоит убедиться, что Retain память ~1KB не переполнена (у нас немало переменных Retain). По расчету, однако, <200 байт, так что запас есть.
15. **Производительность контроллера:** Архитектура, как описана, довольно насыщенная (PID-регулятор, обмен Modbus Master+Slave, ведение логов). ПР102 должен справиться, но важно правильно расставить приоритеты выполнения (например, PID и управление – в цикле 100 мс, а обмен и логи – в более медленном). Предполагаем, что OwenLogic позволяет разделить по таскам или использовать прерывания по таймеру для критичных частей (PID).
16. **Проверка на соответствие v1.55:** В целом, структура соответствует описанию версии ПО 1.55, вариант1. Все опции с 1 по 17 учтены, за исключением тех, что не влияют на программу (настройка типа датчика – это настройка AI, делается через меню контроллера). Основной риск рассогласования – тонкости CIP (см. п.1-2 выше) и взаимодействие с HMI (п.7,10). Требуется совместно с технологом и специалистом по HMI скорректировать эти детали при реализации.

Несмотря на перечисленные вопросы, предложенная архитектура является однозначной, модульной и готовой к непосредственной реализации в OwenLogic. Она охватывает все режимы управления ВДП, обеспечивает требуемый обмен с SCADA и учитывает расширяемость под различные конфигурации оборудования. После прояснения открытых пунктов можно переходить к программированию контроллера и конфигурированию HMI с использованием подготовленных таблиц переменных и CSV-файлов.