**Введение:** Данный документ описывает архитектуру и функционал ПО пастеризатора на базе ПЛК ОВЕН ПР102 (проект OwenLogic). Рассмотрены все предоставленные исходники (файлы ST для функциональных блоков) и таблицы переменных. Приведены четкие требования к работе системы, интерфейсам, взаимодействию блоков, а также выявлены проблемные места текущей реализации. Документ служит ТЗ версии 0.1 для доработки кода – после реализации указанных правок программа должна запускаться и функционировать корректно «с первого раза».

**Сводка обнаруженных разрывов и необходимых доработок:** *(детали – в разделах ниже)*

* **Несоответствия типов и интерфейсов:** выход PasteurManager HeatingPowerReq реализован как BOOL (разрешение нагрева), тогда как вход ActuatorManager Pasteur\_HeatPowerReq ожидает REAL【114†】. Требуется унификация – например, сделать оба как BOOL Pasteur\_HeatEnable либо реализовать процентную команду нагрева как REAL на обоих. Также переменные PV\_TempProduct\_Raw, PV\_TempJacket\_Raw, SP\_DoseVolume ошибочно обозначены как Boolean в таблице, фактически должны быть REAL (°C, л)【92†】. Эти несоответствия следует исправить во всех местах (код и таблицы).
* **Неиспользуемые/неинициализированные сигналы:** Ряд глобальных переменных никогда не меняется программой. Например, STAT\_MixerRunning (флаг работы мешалки) нигде не устанавливается – нужно обновлять его (например, на основании статуса ПЧ или включения DO мешалки). STAT\_HeatTimeout (флаг превышения лимита нагрева) не используется – необходимо задействовать при превышении SP\_HeatTime либо удалить. Флаг Hyst\_HeatRequest сейчас не формируется (ранее формировался во внешней логике) – требуется генерировать внутри TemperatureManager при отключенном PID. Несколько резервных/неиспользуемых сигналов (ALM\_Reserved\*, MODE\_Request и др.) помечены как [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ] – их целесообразность уточняется.
* **Логика “single-writer” нарушена или требует жесткой фиксации:** Все физические выходы DO/AO должны управляться **только** из FB\_ActuatorManager – в текущем коде это выдержано (прямых присваиваний DO/AO вне ActuatorManager нет【40†】). Однако, разрешение нагрева (Heaters) должно поступать **только** от TemperatureManager; в коде PasteurManager фактически управляет нагревом через переменную HeatingPowerReq, что дублирует функцию TemperatureManager. Требуется перераспределить ответственность: PasteurManager лишь задаёт уставку и разрешение нагрева (EN\_Heat), а TemperatureManager выполняет ПИД/гистерезис и управляет нагревателями. Любые попытки управления нагревателями (DO6–DO8) минуя TemperatureManager запрещены. Аналогично, **любой** выход DO/AO должен иметь единственного писателя – FB\_ActuatorManager; проверка кода не выявила случаев обхода, что должно быть зафиксировано как инвариант.
* **Масштабирование аналоговых сигналов:** Проверка выявила потенциальную ошибку масштабирования AO\_1 (нагрев). Переменная AO\_HeaterPower хранится в **0..1000** (что соответствует 0–100%), однако ActuatorManager присваивает AO\_1 = Temp\_AO\_HeaterPower напрямую【120†】, где Temp\_AO\_HeaterPower генерируется в диапазоне 0..100 (%). Таким образом, на выход AO\_1 сейчас подается 0–100 (из 1000) – лишь 10% диапазона. Необходимо либо менять масштаб AO\_HeaterPower на 0..100, либо домножать выход Temp\_AO\_HeaterPower на 10 (0–1000) перед выдачей на AO\_1. В ТЗ принято: **AO\_HeaterPower** (глобальный) 0–1000, TempManager выдает 0–100 (% → далее масштабирует ActuatorManager). Аналогично проверить и унифицировать масштаб задатчика частоты мешалки (0..1000 = 0..100%), параметров температуры (\*0.1 °C) и т.д.
* **Приоритеты режимов и блокировок:** Требуется зафиксировать матрицу взаимного исключения режимов: ручной режим имеет наивысший приоритет (активен – блокирует автоциклы); среди автоматических – CIP выше пастеризации, пастеризация выше режима розлива. В текущем коде это частично реализовано через переменную ActiveMode\_UD【47†】【48†】. Следует описать точные правила: одновременный запрос нескольких режимов – как выбирается активный; переходы между режимами; что запрещено включать одновременно. Также необходимо устранить возможность забыть отключить нагрев при переключении режима (например, если пастеризация прервана и начат CIP – убедиться, что EN\_Heat снят).
* **Watchdog и сбросы команд:** Интерфейсы обмена (HMI, ПЧ) требуют обработки таймаутов. Переменные Heartbeat (MB\_HMI\_Heartbeat, MB\_PLC\_Heartbeat) есть, но не описано, что делать при их остановке – нужно реализовать сброс команд и переход в безопасное состояние (например, останов всех выходов, если связь с HMI потеряна более чем на n сек – [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ], либо хотя бы зафиксировать в MB\_CommFlags). Аналогично с ПЧ: STAT\_VFD\_Online есть, но логику потери связи/ошибки ПЧ надо прописать (например, если нет обмена – останавливать мешалку, показывать аварию). Кроме того, HMI-команды MB\_CmdMask\* после обработки должны сбрасываться – в коде не найдено явного сброса бит Modbus-команд, что может приводить к повторному срабатыванию. Нужно добавить очистку масок команд MB\_CmdMask1/2 после фиксации фронтов (либо использовать механизмы момента).
* **Прочие доработки:** Необходимо добавить недостающие взаимоблокировки: например, если сработал аварийный уровень воды в рубашке (DI3\_LevelJacket=0), то блокировать нагрев (установить BLOCK\_HeaterOff=TRUE) и выдать аварию ALM\_NoWaterJacket. Сейчас ALM\_NoWaterJacket есть, но где устанавливается – не очевидно, нужно задействовать. Также убедиться, что при E-Stop (DI7=0) мгновенно останавливаются все выходы (можно посредством BLOCK\_AllOff). В коде E-Stop не просматривается явно – добавить логику: при DI7=0 задать BLOCK\_AllOff. В режиме ручном необходимо явно указать, что он “прерывает” автоциклы – при входе в ManualActive нужно останавливать запущенный цикл (Pasteur/CIP/Dosing) и сбрасывать их RunReq, иначе выходы просто переразовьются (сейчас ActuatorManager просто по приоритету берет Manual, но внутренние состояния циклов могут остаться “активными”). Эти и другие моменты отражены далее в тексте.

<details><summary>Оглавление</summary>

* **1. Архитектура ПО и режимы работы**
* 1.1 Структурная схема и модули FB
* 1.2 Матрица режимов (Manual / CIP / Pasteur / Dosing)
* 1.3 Обмен данными с HMI (Slot1) и ПЧ (Slot2), мониторинг связи
* **2. Интерфейсы и функционал модулей (паспорт FB)**
* 2.1 FB\_Main\_PR102 – основной цикл
* 2.2 FB\_ActuatorManager – управление выходами (DO/AO)
* 2.3 FB\_TemperatureManager – регулирование температуры
* 2.4 FB\_PasteurManager – цикл пастеризации
* 2.5 FB\_CIPManager\_v2 – цикл CIP-мойки
* 2.6 FB\_DosingManager – режимы розлива
* 2.7 FB\_VFDManager – управление частотным приводом мешалки
* 2.8 FB\_SoftFlowmeter – программный расходомер
* **3. Словарь сигналов (общие переменные)**
* 3.1 Дискретные входы/выходы (DI/DO)
* 3.2 Аналоговые входы/выходы (AI/AO)
* 3.3 Сетевые регистры обмена (Modbus)
* 3.4 Внутренние флаги и параметры
* **4. Правила и инварианты системы**
* 4.1 “Единственный писатель” для выходов и управляющих сигналов
* 4.2 Приоритеты режимов и арбитраж команд
* 4.3 Безопасные состояния (потеря связи, аварии)
* 4.4 Фильтрация, гистерезис, защита ΔT
* 4.5 Именование, области ответственности, исключение дублей
* **5. План тестирования и критерии приёмки**
* 5.1 Тесты без подключенного оборудования (моделирование)
* 5.2 Тесты на стенде/оборудовании
* 5.3 Чек-лист сценариев и ожидаемых результатов
* **6. Перечень доработок и патчей кода**
* 6.1 Изменения в FB\_Main\_PR102
* 6.2 … в FB\_ActuatorManager
* 6.3 … в FB\_TemperatureManager
* 6.4 … в FB\_PasteurManager
* 6.5 … в FB\_CIPManager\_v2
* 6.6 … в FB\_DosingManager / FB\_SoftFlowmeter
* 6.7 … в FB\_VFDManager
* 6.8 План переименования (alias-plan)
* **Приложения:**
* A. **Signals\_Dictionary.csv** – полный словарь сигналов (имя, тип, адрес, единицы, описание).
* B. **XREF.csv** – кросс-референс (модули записи/чтения по каждой переменной, проблемные места).
* C. **DO\_AO\_Tree.csv** – таблица формирования выходов.
* D. **Masks\_BitMap.csv** – расшифровка битовых масок MB.
* E. **Предложение по изменению имён** – соответствие старых/новых, временные «мосты».
* F. **Листинг патчей ST** – фрагменты кода для вставки по файлам.

</details>

## 1. Архитектура ПО и режимы работы

**1.1 Структурная схема и модули FB:** Логика управления разбита на несколько функциональных блоков (FB), взаимодействующих через глобальные переменные (область Ram) и сетевые регистры. Ниже описаны основные модули и их связь (в скобках указаны имена исходников):

* **FB\_Main\_PR102** – главный цикл программы, опрос входов/Modbus, арбитраж режимов и вызовы менеджеров. Является точкой интеграции всех остальных блоков (подробно в 2.1).
* **FB\_ActuatorManager** – менеджер исполнительных устройств. Получает команды от других модулей (пастеризация, CIP, розлив, регулятор температуры, ручной режим) и непосредственно управляет физическими **DO1–DO14** и **AO1–AO2** выходами ПР102【30†】【36†】. Обеспечивает приоритет ручного управления, запрещающие блокировки, единый “слой абстракции” над выводами (см. 2.2).
* **FB\_TemperatureManager** – регулятор температуры продукта. Выполняет преобразование и фильтрацию сигналов с датчиков температуры, ПИД-регулирование (либо двухпозиционный режим) нагрева по заданной уставке, управление группами нагревателей и/или аналоговым клапаном пара. Выдает команды нагрева (DO6–DO8, AO1) через ActuatorManager. Следит за превышением ΔT между рубашкой и продуктом, минимальным временем включения/выключения, авариями датчиков (см. 2.3).
* **FB\_PasteurManager** – управляет автоматическим циклом пастеризации продукта. Стадии: разгон до температуры пастеризации, выдержка, охлаждение (и при необходимости – режим хранения). Формирует команды на подсистемы: включает циркуляционный насос рубашки, мешалку продукта, открывает/закрывает клапан охлаждения, дает разрешение на нагрев на этапах Heat/Hold. Отслеживает таймер выдержки, лимит времени нагрева и аварийные условия (см. 2.4).
* **FB\_CIPManager\_v2** – управляет CIP мойкой по заданной программе. Позволяет выполнять до 7 циклов (этапов) мойки, каждый из которых состоит из фаз подачи сред (хол./гор. вода, щёлочь, кислота), циркуляции и дренажа【81†】. Различные режимы CIP (Full, Alk, Acid, Rinse) представляют преднастроенные маски выбора циклов. CIPManager координирует открытие соответствующих клапанов CIP (DO9–DO12, DO14), работу насоса CIP (DO13), при необходимости – мешалки, контролирует таймеры фаз и общую длительность, обрабатывает прерывание/останов мойки (см. 2.5).
* **FB\_DosingManager** – управляет режимами розлива продукта: **непрерывным** (ручное включение/выключение насоса) и **порционным** (автоматическая дозировка заданного объема). Отслеживает датчики “пистолета” и кнопки порционного розлива, управляет насосом розлива (DO5) через ActuatorManager. Для определения объема использует программный “расходомер” SoftFlowmeter (см. 2.6).
* **FB\_VFDManager** – отвечает за управление частотным приводом (ПЧ) мешалки при **PAR\_UseVFD=TRUE**. Формирует управляющее слово и слово задания частоты для ПЧ (через Modbus Slot2), на основе команд «Пуск/Стоп/Реверс мешалки» и заданной скорости с HMI. Контролирует статус ПЧ (слово статуса, авария)【56†】【57†】. При **PAR\_UseVFD=FALSE** модуль находится в пассивном режиме (выходы неактивны, управление мешалкой идет по аналоговому сигналу) – см. 2.7.
* **FB\_SoftFlowmeter** – вычисляет перекачанный объем продукта по времени работы насоса и заданной производительности (л/с). Используется DosingManager для порционного розлива. Inteгратор имеет вход Enable, который открывается при работе насоса, считает объем, и сбрасывается командой сброса (см. 2.8).

Все модули, кроме Main, реализованы как Funciton Block, но в данном проекте каждый FB вызывается ровно один раз (в цикле MAIN) и оперирует глобальными переменными. Таким образом, связь между блоками – **через глобальные переменные Ram**. Например, PasteurManager.VarOutput CoolingValve\_ON записывает глобальный флаг, который считывает ActuatorManager.VarInput Pasteur\_CoolingValve\_ON【30†】для управления DO1. Аналогично, DosingManager.VarOutput FillPump\_ON записывает глобальный Dosing\_FillPump\_ON, который читает ActuatorManager и включает DO5 при активном режиме розлива.

Ниже представлена обобщенная схема взаимодействия модулей и сигналов:

*Рис.1: Структурная схема ПО пастеризатора.* На схеме показаны основные модули и направления обмена: синим – команды от менеджеров режимов к ActuatorManager, оранжевым – данные от датчиков/HMI к менеджерам, зеленым – обратные связи и статусы.

**1.2 Матрица режимов (Manual / CIP / Pasteur / Dosing):** Программой предусмотрено несколько режимов работы, взаимоисключающих друг друга на уровне управления процессом. Ниже приводится матрица приоритетов и совместимости:

* **Ручной (Manual)** – высший приоритет. В этом режиме оператор напрямую управляет выходами с панели (битовые команды Man1..Man14). При активном ручном режиме все автоматические циклы должны быть отключены/приостановлены. В коде реализовано как флаг Mode\_Manual\_ACTIVE, при котором ActuatorManager игнорирует автоматические команды и копирует ручные сигналы на выходы【41†】【42†】. *Примечание:* При входе в ManualActive следует сбрасывать активность Pasteur/CIP/Dosing (если были включены) – в текущем коде этого нет, нужно доработать (см. 6.x).
* **CIP-мойка** – следующий приоритет после ручного. Не может выполняться одновременно с пастеризацией или розливом, т.к. использует общие ресурсы (емкости, коммуникации). Если CIP запущен (любого типа), запросы пастеризации/розлива игнорируются до завершения CIP. В реализации, Main присваивает ActiveMode\_UD значения 10–13 при запросе CIP (Full=10, Alk=11, Acid=12, Rinse=13) с приоритетом перед Pasteur (1) и Dispense (2,3)【49†】. Это гарантирует, что при одновременном нажатии, например, Start CIP и Start Pasteur, будет выбран CIP. Другие режимы ждут окончания CIP (или принудительного стопа CIP). В ActuatorManager флаг Mode\_CIP\_ACTIVE управляет выходами CIP: при TRUE активны команды CIPManager (например, DO9–DO14) и блокируются другие (охлаждение DO1, циркуляция DO2 и др., см. 2.2).
* **Пастеризация** – имеет приоритет ниже CIP, выше режимов розлива. Выполняется, если не активны CIP и Manual. При старте пастеризации (Mode\_Pasteur\_RunReq) проверяются блокировки (наличие продукта, датчики, не активен ли CIP). Во время пастеризации могут быть параллельно включены только подрежимы: выдержка (hold) или охлаждение, но **не** режим розлива (розлив начинается после полного окончания пастеризации или в отдельном режиме Dispense). В ActuatorManager флаг Mode\_Pasteur\_ACTIVE позволяет командам PasteurManager управлять выходами (DO1, DO2, DO3 и т.д.)【42†】. Если во время пастеризации поступит запрос CIP, он будет поставлен в очередь (или пастеризация прервется по аварийному условию – см. ALM\_PasteurAbort).
* **Розлив (Dispense)** – наименьший приоритет из автоциклов. Возможен *только* когда не активны Pasteur или CIP (и Manual). Режим розлива может быть двух видов: непрерывный или порционный. Реализовано двумя командами: Mode\_DispCont\_RunReq и Mode\_DispDose\_RunReq (взаимоисключающие). При активном Dispense задействуются свои выходы (насос розлива, при необходимости мешалка – но автоматика мешалку не включает, [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ], возможно для хранения продукта потребуется вращать мешалку вручную или через опцию Store). Если во время розлива оператор запускает пастеризацию или CIP, то розлив должен быть остановлен (в коде Main это обеспечено: при запросе Pasteur/CIP ActiveMode переключится, и DosingManager не получит RunReq, выключив насос). Приоритет Pasteur над Dispense подтверждается логикой ActiveMode: Pasteur (1) назначается если нет CIP, иначе Dispense (2 или 3)【47†】. Таким образом, одновременный запуск пастеризации и розлива – розлив не начнется до окончания пастеризации.

**Таблица 1. Матрица приоритетов и запретов режимов:**

| Активный режим \ Запрошен | Manual | CIP (Full/Alk/…) | Pasteur | Dispense (Cont/Dose) |
| --- | --- | --- | --- | --- |
| **Manual** | – | **блокирует CIP**<br>(игнорировать)【41†】 | **блокирует Pasteur** | **блокирует Dispense** |
| **CIP** | **прерывает Manual?**<br>*не допускается* | – | **запретить Pasteur**<br>до стоп CIP | **запретить Dispense** |
| **Pasteur** | **откл. Manual**<br>(или не входить) | **отложить CIP**<br>до стоп Pasteur | – | **запретить Dispense** |
| **Dispense** | **откл. Manual** | **отложить CIP** | **отложить Pasteur** | – (Cont vs Dose взаимоискл.) |

Примечания: - В Manual оператор может принудительно включать любые механизмы, даже если автоцикл ранее работал. Это означает потенциально конфликтующие ситуации (например, в середине пастеризации включен Manual – нагрев останется активным через ручной бит). Поэтому **при входе в Manual** рекомендуется: а) установить Mode\_Manual\_ACTIVE=TRUE, б) обнулить флаги Mode\_PasteurActive/Mode\_CIPActive/Mode\_DispenseActive (прервать автоциклы), в) перевести все выходы в ручное управление (в коде ActuatorManager это делается при каждом цикле). После выхода из ручного режима (если такое предусмотрено – обычно Manual toggled off) нужно решить, возвращаться ли в прерванный цикл (не рекомендуется) или оставить систему в Idle.

* CIP, Pasteur, Dispense – **режимы Auto**. В системе предусмотрен флаг MB\_SystemStatus (битовая маска) с битом “Manual” (b0) и “Auto” (b1). Принято: если любой из автоциклов активен (Pasteur/CIP/DispenseActive = TRUE), то система считается в автоматическом режиме (SystemStatus.b1=1). Если ни один не активен – система Idle (ожидает команд). ManualActive при этом =0. Биты SystemStatus формируются в Main на основании активных режимов.

**1.3 Обмен данными с HMI и VFD, контроль связи:**

ПЛК ПР102 оснащен двумя портами RS-485 (Slot1 – ведущий Modbus RTU для HMI ОВЕН; Slot2 – ведущий Modbus RTU для частотника AD30). Обмен осуществляется через глобальные переменные, соответствующие Modbus-регистрам, с префиксом MB\_ (для HMI) и VFD\_ (для ПЧ). Ниже перечислены ключевые регистры и их назначение:

* **Команды HMI (MB\_CmdMask1, MB\_CmdMask2):** Два регистра типа UDINT (по 16 бит каждая маска) используются для команд оператора【10†】. MB\_CmdMask1 – команды запуск/останов автоциклов и вспомогательные:
* **b0** – Start Pasteur (запуск пастеризации);
* **b1** – Stop Pasteur (преждевременная остановка пастеризации);
* **b2** – Skip Heat (пропуск стадии нагрева – переход сразу к выдержке)【10†】;
* **b3** – Reset Alarms (сброс аварийных сигналов)【10†】;
* **b4** – Start Continuous Dispense; **b5** – Stop Continuous Dispense;
* **b6** – Start Dose Dispense; **b7** – Stop Dose Dispense;
* **b8** – Next Dose (подача следующей порции вручную);
* **b15** – Reset SoftFlowmeter (сброс счетчика наливов). Маска MB\_CmdMask2 – команды ручного управления и CIP:
* **b0** – Mixer motor Start/Stop (в авто ручной режим мешалки);
* **b1** – Heater group1 on/off; **b2** – Heater group2 on/off; **b3** – Cool valve on/off;
* **b4** – Mixer FWD; **b5** – Mixer REV (регистрируются, если оборудование поддерживает реверс; наш ActuatorManager их обрабатывает как два отдельных DO3/DO4);
* **b6** – Start CIP Full; **b7** – Start CIP Alk; **b8** – Start CIP Acid; **b9** – Start CIP Rinse;
* **b10** – Stop CIP (прерывание CIP цикла). ПЛК отслеживает фронты этих бит и формирует соответствующие запросы режимов (reqPasteur, reqCIP\_Full и т.п.). После обработки **ПЛК должен сбрасывать биты** (чтобы HMI видела исполнение команды). *В текущей реализации, сброс бит пока не реализован – необходимо добавить: после фиксации команды, обнулить соответствующий бит Mask (сделать MB\_CmdMask1.x:=0).*
* **Параметры HMI (MB\_ParMask):** Регистр MB\_ParMask содержит флаги настроек (параметров), которые оператор может включать/выключать【13†】. Разрядность 32 бита; используются:
* **b0 UsePID** – использовать ПИД-регулятор (если 0 – работать в режиме гистерезиса);
* **b1 UseVFD** – использовать управление мешалкой через ПЧ (0 – аналоговое);
* **b2 UsePortionButton** – запускать порцию только по внешней кнопке (DI4) в режиме дозированного розлива;
* **b3 UseGunSensor** – разрешать розлив только при надетом «пистолете» (датчик на месте, DI5);
* **b4 HeaterGrp2** – разблокировать группу нагрева №2 (если 0 – не использовать DO7 даже если оборудование есть);
* **b5 HeaterGrp3** – разблокировать группу нагрева №3;
* **b6 StoreEnabled** – режим хранения (поддержание температуры после пастеризации) разрешен;
* **b16 PortionDone** – (возможно, зарезервирован под сигнал окончания порции – требует уточнения, т.к. PortionDone у нас отдельной переменной). Эти биты ПЛК читает и в основном использует как параметры (PAR\_UsePID, PAR\_UseVFD и т.д.). Обратно ПЛК может выставлять некоторые – например, b16 PortionDone можно устанавливать на 1 на короткое время по завершении дозы (для индикации на HMI).
* **Выходные статусы HMI:**
* MB\_OutStateMask (UDINT) – маска состояний выходов ПЛК. Каждому физическому DO соответствует бит (0 – DO1, 1 – DO2, … 13 – DO14). ПЛК формирует эту маску каждый цикл: если выход включен, соответствующий бит =1. На HMI это отображает, какие механизмы сейчас активны. *(Формируется в Main: через битовые сдвиги или побитовое ИЛИ всех DO…)*
* MB\_Alarms (UINT или UDINT) – маска активных аварий. Используются биты 0–15 (см. список ALM\_\* в разделе 3.4). ПЛК собирает эту маску из флагов ALM\_xxx (например, если ALM\_SensorProduct\_Break=TRUE – устанавливает бит0). Маска читается HMI (регистр 512).
* MB\_SystemStatus (WORD) – статус системы: b0 “Manual mode active”, b1 “Auto mode active”, b2 “Есть активная авария” (Alarm Active), b3 “Alarm reset required/pressed”. ПЛК выставляет б0/b1 по активности режимов, b2 – если любой ALM\_\*=TRUE, b3 – при нажатии ResetAlarms (можно мигать, чтобы показать запрос).
* MB\_CIPStagesMask и MB\_PasteurStagesMask (WORD каждая) – маски этапов CIP и пастеризации. *Назначение:* при выборе режима CIP или Pasteur, ПЛК заносит сюда, какие стадии входят в цикл. Например, при CIP Full MB\_CIPStagesMask = 0x7F (127) – значит биты 0-6 = 1, т.е. циклы 1–7 будут выполняться. Для CIP Alk маска = 103 (0x67) – биты 0,1,2,5,6=1 (т.е. циклы 1,2,3,6,7 активны – пропущены 4,5). Для Acid = 115 (0x73), Rinse = 96 (0x60). Для пастеризации MB\_PasteurStagesMask по сути всегда одна “маска” (например, биты 0=Heat, 1=Hold, 2=Cool) – [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ]. Вероятно, тут отображается текущая стадия пастеризации, или входящие стадии (например если skip heat – то убирается бит0). Нужно уточнить логику.
* STEP\_Pasteur, STEP\_CIP, STEP\_Dispense (Word, рег.545-547) – коды текущего шага/этапа для отображения на HMI. Например, Pasteur: 0=Idle, 1=Нагрев, 2=Выдержка, 3=Охлаждение, 4=Готово. CIP: номер активного цикла 1–7, Dispense: состояние дозатора (0=остановлен, 1=льет непрерывно, 2=льет порцию, 3=порция завершена и ждет). Эти коды задаются соответствующими менеджерами.
* **Параметры уставок с HMI:** множество регистров типа SP\_\* или MB\_\* служат для задания уставок температур и времен:
* Температура пастеризации SP\_HeatTemp\_x10 (рег.523) – задается в 0.1 °C. ПЛК читает и преобразует в REAL °C для PasteurManager.SP\_HeatTemp【76†】. Граница допустимых значений ограничивается константами (например, 50.0–150.0 °C для продукта).
* Ограничение времени нагрева SP\_HeatTime\_min (рег.524) – задается в минутах. PasteurManager использует для аварийного флага STAT\_HeatTimeout – если нагрев не вышел на температуру за заданное время (но сейчас не реализовано).
* Температура выравнивания SP\_SmoothTemp\_x10 (525) и дифференциал выравнивания SP\_SmoothDiff\_x10 (526) – параметры “плавного разгона”. **В данной версии не используются напрямую для нагрева**, но SP\_SmoothDiff применяется как гистерезис при охлаждении (см. 2.4 PasteurManager) – он задает допуск по температуре рубашки ниже уставки охлаждения, при котором закрывается клапан охлаждения【79†】.
* Время выдержки SP\_HoldTime\_s (527) – задает длительность пастеризации (выдержки) в секундах. Используется PasteurManager (таймер hold).
* Температура охлаждения SP\_CoolTemp\_x10 (568) – конечная температура после охлаждения, в 0.1 °C. Используется PasteurManager для этапа охлаждения (клапан охлаждения открывается пока продукт выше этой температуры).
* **Параметры CIP:** регистры 530–533 MB\_CIP\_Edit\_\* – используются для редактирования рецепта CIP. Согласно комментарию【8†】: это уставки длительностей на этапе 1 (Cycle1) – T\_COLD, T\_ALK, T\_ACID, T\_PUMP (в секундах) для выбранного цикла. Похоже, реализован механизм: оператор выбирает цикл (например, Cycle3) и меняет значения этих 4 регистров, затем по истечении CIP\_LiveWriteCommitDelay\_s (параметр, по умолчанию 2с) CIPManager переносит эти значения в свои retain-переменные CIP\_C3\_T\_\* и сбрасывает флаг редактирования. Также есть MB\_CIP\_Edit\_Select (вероятно рег. 529 или 569) – номер редактируемого цикла [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ]. В целом, CIPManager поддерживает on-line изменение уставок, сохраняя их (ретентивные CIP\_C#\_\*).
* **Другие CIP параметры:** CIP\_DrainOnStopSec (ретентивное, сек) – сколько секунд держать клапан слива открытым после корректного окончания цикла (по умолч. 30с). CIP\_ChemConflictPolicy (0/1/2) – политика разрешения конфликта щелочь+кислота. Предположительно: 0 – запрет запуска CIP, если предыдущий цикл был с другим реагентом (требовать промежуточного ополаскивания); 1 – приоритет ALK (если запущены оба – выполнить Alk, проигнорировать Acid); 2 – приоритет Acid. *Надо уточнить применение.*
* **Mixer parameters:** MB\_SetMixerFreq\_0\_1pct50 (рег.518) – заданная скорость мешалки, 0–1000 соответствует 0–100.0% от 50 Гц. Если PAR\_UseVFD=TRUE, этот регистр используется VFDManager для расчета слова частоты ПЧ. Если PAR\_UseVFD=FALSE (аналоговый привод), то планировалось использовать его для AO2 (например, 500 = 50% = ~5 В), однако текущая реализация ActuatorManager выставляет AO2 просто 100% при включении мешалки без учета этой уставки – это ограничение. *Рекомендуется*: при аналоговом управлении мешалкой учитывать MB\_SetMixerFreq и PAR\_MaxMixerSpeed (максимальная скорость, рег.536) – например, AO2 = (SetFreq \* MaxSpeed / 1000).
* **Dosing parameters:** PAR\_PumpFlowRate (Float, retain) – производительность насоса розлива, л/с. Используется в SoftFlowmeter для расчета объема. SP\_DoseVolume (Float, non-retain) – объем дозы (л). Устанавливается оператором (в HMI, хотя не указано отдельно в сетевых – вероятно в панели привязан напрямую к переменной). Применяется DosingManager как целевой объем на порцию.
* **Обмен с частотным приводом (Slot2):** ПЧ типа OWEN AD30 (или аналог) управляется по Modbus. Файл PR102\_Сетевые\_Слот\_2\_Master\_AD30\_v4.csv содержит регистры:
* VFD\_CmdWord (Addr 8192) – командное слово ПЧ【15†】. Кодируются команды: 1=FWD пуск, 2=REV пуск, 5=CoastStop (моментальный стоп), 6=DecelStop (торможение), 7=ResetFault, 8=ResetFault (оба 7 и 8 сброс аварии)【18†】. Формируется VFDManager в выходе VFD\_CmdWord как UDINT, но фактически используется 16 младших бит (отправляется функцией 0x06 – preset single reg).
* VFD\_SetFreqWord – слово задания частоты. В конфигурации доступны два варианта: **4096**: -10000..10000 (–100..+100% от уставки P0-14) и **36864**: 0..P0-14 с шагом 0.01Гц【18†】. По умолчанию реализовано через 4096 (0x1000) – VFD\_SetFreqPct (–10000..10000) – но нам нужен только положительный диапазон 0..10000 = 0–100% (т.к. реверс командой). VFDManager вычисляет это слово исходя из MB\_SetMixerFreq (0–1000 → 0–100.0%). Например, если MB\_SetMixerFreq=500 (50%), а P0-14 (макс частота) = 50.00 Гц, то VFD\_SetFreqWord = 5000 (~50.00% от 50Гц = 25Гц). В коде VFDManager выход VFD\_SetFreqWord UDINT, ожидается 0..50000 для 0–50Гц【57†】. Значение >100% не используется (если PAR\_MaxMixerSpeed <100%, логика должна ограничить SetFreq).
* VFD\_Status (Addr 12288) – слово статуса ПЧ【18†】. Комментарий: “1=FWD,2=REV,3=STOP (0x3000) [READ-ONLY]” – похоже, в AD30 рег.0x3000 выдает статус вращения (1 или 2 – вращается, 3 – остановлен). VFDManager читает его (вход VarOutput VFD\_Status обновляется Main или самим VFDManager через чтение 0x03 функции). По статусу VFDManager выставляет VFD\_Fault (BOOL) – как именно определяется “авария” не указано, возможно, сравнивая с кодами ошибок, или если нет связи. В AD30, вероятно, есть отдельный регистр аварий – стоит подумать.
* **Heartbeat и связь:** Для Slot2 явного “heartbeat” нет, но можно использовать таймер: если VFD\_Status не обновлялся n секунд – считать связь потерянной. Переменная STAT\_VFD\_Online (Ram) должна отражать успешный обмен (в Main можно обнулить ее если несколько запросов к ПЧ подряд не отвечены). В проекте STAT\_VFD\_Online декларирован, но код установки не найден – надо реализовать. Если Online=FALSE – ActuatorManager/TemperatureManager должны отключить мешалку (например, блокировать DO3/DO4 и не пытаться командовать ПЧ).
* **Прочие регистры мониторинга:** В CSV Slot2 приведены некоторые мониторинги: VFD\_Mon\_SetHz, VFD\_Mon\_RunHz, VFD\_Mon\_DCBus, VFD\_Mon\_OutVolt, VFD\_Mon\_OutCurrent – эти параметры могут читаться ПЛК (0x03) и при необходимости отображаться на HMI, но в нашем ПО они не используются активно (можно добавить в отображение статуса привода).

**Контроль связи (Watchdog):** Как уже упоминалось, необходимо предусмотреть реакции: - **Потеря связи HMI:** можно контролировать через обмен heartbeat: HMI каждую секунду инкрементирует MB\_HMI\_Heartbeat (рег.543), ПЛК – MB\_PLC\_Heartbeat (544)【11†】. Если счетчик HMI не меняется >3 сек – значит панель не отвечает. Действия: либо просто установить бит в MB\_CommFlags (например, bit0=HMI\_Comm\_Lost), либо в безопасное состояние – т.к. HMI в основном для управления, ПЛК может продолжать цикл без панели. Однако, **важно:** если команда на останов могла быть подана только с HMI (например, Stop Pasteur), а связи нет, то оператор не может штатно остановить – ПО должно предусмотреть аварийное прекращение по таймауту панели? В этом проекте критично только аварийный стоп с концевика (DI7) – он сработает вне зависимости от HMI. Поэтому можно ограничиться индикацией и, скажем, блокировкой новых запусков режимов, если HMI offline. [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ политики]. - **Потеря связи ПЧ:** если STAT\_VFD\_Online=FALSE, нужно остановить мешалку. В ручном режиме – не включать DO3/DO4 (ActuatorManager должен игнорировать Man3/Man4 при Offline, или выдавать Alarm). Добавить аварию ALM\_MixerFault (бит4), хотя сейчас ALM\_MixerFault скорее привязан к DI1 (ошибка от частотника/пускателя). Возможно, ALM\_MixerFault будет устанавливаться при срабатывании DI1 (НЗ контакт защитного реле мешалки) **ИЛИ** при сигнале Fault от ПЧ. В VFDManager VFD\_Fault (BOOL) формируется по статусу ПЧ – его нужно использовать: если TRUE, то выставлять ALM\_MixerFault. - **Watchdog SoftFlowmeter:** если в порционном режиме долго нет импульса расходомера (например, датчик не считает, а насос работает) – у нас “расходомер” программный, поэтому эта ситуация не отслеживается. Однако, если бы был импульсный счетчик, стоило бы контролировать (не используется). - **Пропадание датчиков температуры:** это контролируется внутри Main: если значение AI выходит за диапазон (-50…150 °C), то ALM\_Sensor… выставляется【68†】【69†】. Реакция – TemperatureManager должен остановить нагрев (EN\_Heat=FALSE) и PasteurManager должна прервать цикл (с аварией). У нас есть ALM\_PasteurAbort (бит13) – видимо, ставится при критических ситуациях во время пастеризации, таких как пропажа датчика. Нужно реализовать: если ALM\_SensorProduct\_Break/Short сработал во время PasteurActive – ставить ALM\_PasteurAbort и переходить в аварийное завершение цикла (напрямую к охлаждению или останов с сигнализацией). Аналогично для CIPAbort (бит14) – если оператор нажал Stop CIP или датчик потока D6I\_ProductFlow=0 слишком долго. - **Аварийный стоп (E-Stop):** DI7\_EStop (НЗ контакт, 1=норма, 0=авария) – при срабатывании немедленно останавливает все выходы. Это можно выполнить установкой BLOCK\_AllOff=TRUE в Main. ActuatorManager каждую итерацию ставит все DO=FALSE, AO=0, если BLOCK\_AllOff=TRUE (в коде: перед логикой outputs сбрасываются в безопасные значения【40†】, а затем обрабатываются команды только если NOT BLOCK\_AllOff). При нормализации E-Stop (DI7 back to 1) – блокировку можно снимать (но автоциклы не возобновлять без явного запроса заново). - **Нет воды в рубашке:** DI3\_LevelJacket (НЗ, 1=уровень есть, 0=нет) – при 0 следует немедленно отключить нагрев (чтобы не сжечь ТЭНы). В Main можно выставлять BLOCK\_HeaterOff=TRUE, а также ALM\_NoWaterJacket (бит12). В TemperatureManager можно дополнительно принудительно снимать EN\_Heat. В ActuatorManager реализовано: если BLOCK\_HeaterOff=TRUE, то DO6–8 обнуляются независимо от команд【118†】, AO1 также сбрасывается. - **Обратная связь клапанов/механизмов:** В проекте отсутствуют обратные датчики состояний клапанов, кроме DI1/DI2 (аварии мешалки и нагрева). Если были бы, их обработка – аналогично E-Stop: снимать команды, выдавать аварии. Здесь DI2\_HeaterFault (НЗ от термореле) – при 0 нужно отключить нагрев (BLOCK\_HeaterOff) и ALM\_HeaterFault (бит5). DI1\_MixerFault – при 0 отключить мешалку (блок DO3/DO4) и ALM\_MixerFault. Эти DI используются напрямую (НЗ контакты). В ActuatorManager стоило бы добавить: если DI1=0 -> BLOCK\_AllOff или отдельный блокатор мешалки. Но можно проще: DI1/DI2 подключены через внешние реле, обычно они обесточат цепи напрямую, а ALM\_MixerFault/HeaterFault просто индикация на HMI. Тем не менее, укажем: **Инвариант** – при DI1=0 или DI2=0, соответствующее оборудование не включается программно.

*(Раздел 4 ниже формализует многие из этих правил как инварианты системы.)*

## 2. Интерфейсы и функционал модулей (паспорт FB)

В этом разделе для каждого функционального блока приведены: назначение, интерфейс (все входы/выходы с описанием и связями), ключевые алгоритмы (пре-/постусловия, внутренние стадии, таймеры, блокировки). Таблицы интерфейсов содержат колонки: **Signal** (имя сигнала), **Dir** (направление: In=вход FB, Out=выход FB), **Type** (тип данных), **Unit** (единицы измерения/размерность), **Scale** (масштаб или битность), **Source** (источник данных для входа / модуль-потребитель для выхода), **Consumers** (кто использует этот сигнал; для входов – сам FB; для выходов – другие FB или внешнее устройство), **Semantics** (смысл, описание), **Default** (значение по умолчанию/начальное), **Retain** (энергонезависимость).

### 2.1 FB\_Main\_PR102 – основной цикл

**Назначение:** Главный функциональный блок, выполняющий цикл обработки программы. Именно в FB\_Main происходит чтение всех входных сигналов (глобальных DI, AI, регистров Modbus), принятие решений о режиме работы, запуск/остановка соответствующих менеджеров, сбор выходных сигналов и запись глобальных переменных, управляющих оборудованием. FB\_Main также реализует предварительную обработку аналоговых сигналов (температур) и формирование диагностических статусов.

Основные функции: - Определение активного режима работы на основе запросов (бит команд HMI) и текущего состояния (Manual/CIP/Pasteur/Dispense). Установка переменной ActiveMode\_UD и соответствующих флагов Mode\_*RunReq и Mode*Active【47†】【48†】. - Запуск (вычисление входов и вызов) функциональных блоков: PasteurManager, CIPManager, DosingManager, TemperatureManager, VFDManager, ActuatorManager – в нужной последовательности и с нужными параметрами. - Преобразование “сырой” аналоговой информации: перевод AI\_TempProduct/AI\_TempJacket из формата АЦП (UDINT, *0.1°C) в физические REAL °C, с фильтрацией обрыва/КЗ датчиков【68†】. Выдача этих значений в TemperatureManager. - Формирование сводных масок и статусов для HMI: маска аварий ALM\_Mask (битовое ИЛИ всех ALM\_*), MB\_OutStateMask (объединение DO1-DO14)【127†】, MB\_SystemStatus (Manual/Auto/Alarm bits), MB\_CIPStagesMask/PasteurStagesMask (на основе выбираемого режима), код этапа STEP\_\*. - Обработка команд HMI: чтение MB\_CmdMask1/2, формирование внутренних флагов reqXXX, очистка бит команд (при реализации). - Управление блокировками: установка BLOCK\_AllOff при E-Stop или критических авариях, BLOCK\_HeaterOff при нет воды/ошибке нагрева. - Передача глобальных параметров: считывание ParMask (PAR\_UsePID, PAR\_UseVFD и т.п.) и донесение этих флагов до других блоков (например, в VFDManager, ActuatorManager). - Контроль таймеров: в Main заведён секундный тик (Seconds\_UD), инкрементирующийся каждую секунду (может делаться на основе системного RTC или Timer). Он передается в TemperatureManager и SoftFlowmeter для отсчета времени.

**Интерфейс FB\_Main\_PR102:** Основные переменные FB\_Main – глобальные (входы/выходы ПЛК и Modbus) подключены через VAR\_EXTERNAL, но в тексте мы приводим как бы Var\_Input для удобства:

***Входы и основные локальные переменные:***

| Signal | Dir | Type | Unit | Source/Origin | Consumers | Semantics | Default | Retain |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Enable** | In | BOOL | – | Системный запуск блока (из PLC runtime) | FB\_Main (внутр.) | Глобальное разрешение работы FB\_Main (обычно всегда TRUE) | – | – |
| **DI1\_MixerFault** | In | BOOL | – | дискретный вход 1 (НЗ контакт аварии мешалки) | Main, ActuatorMgr | 0 = авария мешалки. Используется для ALM\_MixerFault, блокировки мешалки. | – | – |
| **DI2\_HeaterFault** | In | BOOL | – | дискретный вход 2 (НЗ контакт аварии нагрева) | Main, ActuatorMgr | 0 = авария нагревателя (термостат). Вызывает ALM\_HeaterFault, BLOCK\_HeaterOff. | – | – |
| **DI3\_LevelJacket** | In | BOOL | – | дискретный вход 3 (НЗ датчик уровня рубашки) | Main, TemperatureMgr | 0 = нет воды в рубашке. Приводит к аварии ALM\_NoWaterJacket и отключению нагрева. | – | – |
| **DI4\_PortionBtn** | In | BOOL | – | дискретный вход 4 (НО кнопка порции) | DosingManager | 1 = нажата кнопка подачи порции (используется, если PAR\_UsePortionButton=1). | – | – |
| **DI5\_GunSensor** | In | BOOL | – | дискретный вход 5 (НО датчик “пистолет снят”) | DosingManager | 1 = раздаточный пистолет **снят** (не на месте). Используется если PAR\_UseGunSensor=1: разрешение розлива только при 0. | – | – |
| **DI6\_ProductFlow** | In | BOOL | – | дискретный вход 6 (НО поток продукта/контур CIP) | CIPManager (возможно) | 1 = есть поток (датчик потока). Используется для контроля наличия среды. Пока не задействовано явно. | – | – |
| **DI7\_EStop** | In | BOOL | – | дискретный вход 7 (НЗ аварийный стоп) | Main, ActuatorMgr | 1 = норма, 0 = аварийный стоп нажат. При 0 – немедленное отключение всего (BLOCK\_AllOff). | – | – |
| **AI\_TempProduct** | In | UDINT | 0.1°C | аналоговый вход AI1 (датчик продукта) | Main (преобразует) | Цифровое значение температуры продукта (×10). 0 = 0.0°C, 200 = 20.0°C. Диапазон ~ -50.0…150.0°C. | – | – |
| **AI\_TempJacket** | In | UDINT | 0.1°C | аналоговый вход AI2 (датчик рубашки) | Main (преобразует) | Значение температуры рубашки (×10). Диапазон как выше. | – | – |
| **MB\_Alarms** | In | UINT | битовая маска | Modbus рег.512 от HMI (чтение) | Main (не пишет) | Маска аварий, сохраняемая в панели. **Вход** для ПЛК смысла не имеет (ПЛК формирует ее). | 0 | No |
| **MB\_CmdMask1** | In | UDINT | битовая маска | Modbus рег.541 (команды 1)【10†】 | Main (парсит, сбрасывает) | Команды запуска/остановки режимов (биты b0..b15, см. описание выше). ПЛК считывает фронты. | 0 | No |
| **MB\_CmdMask2** | In | UDINT | битовая маска | Modbus рег.542 (команды 2)【11†】 | Main (парсит, сбрасывает) | Команды ручного управления и CIP (b0..b10). | 0 | No |
| **MB\_ModeCurrent\_in** | In | UDINT | код | Modbus рег.539 (резерв) | – | Резервный вход: предполагалось получать текущий режим с HMI, но не используется (ПЛК сам решает режим). | 0 | No |
| **MODE\_Request** | In | UDINT | код | – (возможно, Modbus 540) | – | Еще один резерв: возможно, планировалось использовать как запрос режима (0=Idle,1=Pasteur,...), но реализовано через Mask bits. | 0 | No |
| **MB\_HMI\_Heartbeat** | In | UINT | counter | Modbus рег.543 (HMI→PLC)【11†】 | Main (контроль) | Счетчик heartbeat от панели. Инкрементируется HMI раз в секунду. Если не меняется – потеря связи. | – | No |
| **Seconds\_UD** | Loc | UDINT | секунда | (внутренняя) RTC/таймер | TempMgr, Flowmeter | Секундный таймер (монотонный). Используется для временных вычислений в других блоках. | 0 | No |
| **reqPasteur, reqDispCont, reqDispDose, reqCIP\_Full, reqCIP\_Alk, reqCIP\_Acid, reqCIP\_Rinse** | Loc | BOOL | – | HMI командные биты | Main (internal logic) | Внутренние флаги “запрошен режим”. Становятся TRUE на фронт соответствующей команды Start, остаются до сброса или до отмены. Используются для выбора ActiveMode. | FALSE | – |
| **ActiveMode\_UD** | Loc | INT | код режима | Main (расчет)【49†】 | Main (для раздачи RunReq) | Код выбранного активного режима: 0=Idle/None, 1=Pasteur, 2=DispCont, 3=DispDose, 10=CIP Full, 11=CIP Alk, 12=CIP Acid, 13=CIP Rinse. Определяется приоритетно: сначала проверяются reqCIP\*, затем Pasteur, затем Dispense. | 0 | – |
| **PAR\_UsePID** (from ParMask b0) | Loc | BOOL | – | MB\_ParMask b0【13†】 | TempMgr | 1 = использовать ПИД-регулятор, 0 = гистерезис. Main читает ParMask и передает как вход в TemperatureManager. | 0 (False) | Yes (ParMask retain) |
| **PAR\_UseVFD** (ParMask b1) | Loc | BOOL | – | MB\_ParMask b1【13†】 | ActuatorMgr, VFDMgr | 1 = управлять мешалкой через ПЧ (Modbus), 0 = через аналоговый сигнал/реле. Влияет на логику Actuator (не включает DO3/DO4 при VFD) и активность VFDManager. | 0 | Yes |
| **PAR\_UsePortionButton** (b2) | Loc | BOOL | – | MB\_ParMask b2【13†】 | DosingMgr | 1 = запуск каждой порции только по нажатию внешней кнопки (DI4) или команды NextDose. 0 = порции льются автоматически подряд в режиме Dose. | 0 | No (ParMask no retain, but copied to Ram) |
| **PAR\_UseGunSensor** (b3) | Loc | BOOL | – | MB\_ParMask b3【13†】 | DosingMgr | 1 = разрешать розлив только если “пистолет на месте” (DI5=0). DosingManager при b3=1 не начнет/продолжит налив, пока датчик не замкнут. | 0 | No |
| **PAR\_HeaterGroup2Enabled** (b4) | Loc | BOOL | – | Ram (retain PAR) | TempMgr | 1 = группа нагрева 2 (DO7) доступна, 0 = не использовать DO7 даже если нужна мощность >33%. TempManager проверяет перед включением grp2. | 1 | Yes |
| **PAR\_HeaterGroup3Enabled** (b5) | Loc | BOOL | – | Ram (retain PAR) | TempMgr | 1 = группа 3 (DO8) доступна, 0 = не использовать. | 1 | Yes |
| **PAR\_StoreEnabled** (b6) | Loc | BOOL | – | Ram (retain PAR) | PasteurMgr | 1 = после пастеризации переходить в режим хранения (поддержание температуры). PasteurManager при b6=1 вместо полного остановки после охлаждения может удерживать температуру CoolTemp. | 0 | Yes |
| **… остальные PAR\_XXX** | – | – | – | – | – | Остальные параметры ParMask (UseSteam, PortionDone flag etc.) – обрабатываются по месту, см. 2.3, 2.6. | – | – |

***Выходы (Var\_Output) и основные действия:***

| Signal | Dir | Type | Unit | Source (in Main) | Consumers | Semantics | Default | Retain |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **EnableOut** | Out | BOOL | – | (calc: =Enable AND NOT E-Stop) | ActuatorMgr? | Глобальный выход разрешения работы. Может не использоваться напрямую (зарезервировано). | FALSE | No |
| **PV\_TempProduct\_Raw** | Out | REAL | °C | calc: if AI in range then =AI/10【68†】 | TempMgr (in)【60†】, HMI | Температура продукта, несглаженная, но валидированная. Передается в TempMgr.PV\_TempProduct\_Raw. Также может отображаться на HMI. | 0.0 | No |
| **PV\_TempJacket\_Raw** | Out | REAL | °C | calc: =AI\_TempJacket\_x10/10 | TempMgr (in)【60†】, HMI | Температура рубашки сырая. Передается в TempMgr.PV\_TempJacket\_Raw. | 0.0 | No |
| **HeatingPowerReq\_IN** | Out | REAL | % (0..300) | PasteurMgr (out?) | TempMgr? ActuatorMgr? | ***Неоднозначно***: переменная для передачи запроса мощности нагрева【65†】. Возможно, используется как посредник между PasteurManager.HeatingPowerReq и TempMgr.EN\_Heat или Actuator. Сейчас mismatched. [Будет убрана/заменена на BOOL EN\_Heat]. | – | No |
| **Mode\_Pasteur\_RunReq** | Out | BOOL | – | calc: =(ActiveMode\_UD==1 AND reqPasteur)【47†】 | PasteurMgr (in)【75†】 | Сигнал запуска/останова режима пастеризации. Подается на PasteurManager.RunReq (1 = начать/работает, 0 = остановить цикл). | FALSE | No |
| **Mode\_DispCont\_RunReq** | Out | BOOL | – | =(ActiveMode\_UD==2 AND reqDispCont)【47†】 | DosingMgr (in)【84†】 | Запрос непрерывного розлива. | FALSE | No |
| **Mode\_DispDose\_RunReq** | Out | BOOL | – | =(ActiveMode\_UD==3 AND reqDispDose)【47†】 | DosingMgr (in) | Запрос порционного розлива. | FALSE | No |
| **Mode\_CIP\_Full\_RunReq** | Out | BOOL | – | =(ActiveMode\_UD==10 AND reqCIP\_Full)【48†】 | CIPMgr (in)【83†】 | Запрос CIP полного цикла. | FALSE | No |
| **Mode\_CIP\_Alk\_RunReq** | Out | BOOL | – | =(ActiveMode\_UD==11 AND reqCIP\_Alk)【48†】 | CIPMgr (in) | Запрос CIP-щелочь. | FALSE | No |
| **Mode\_CIP\_Acid\_RunReq** | Out | BOOL | – | =(ActiveMode\_UD==12 AND reqCIP\_Acid)【48†】 | CIPMgr (in) | Запрос CIP-кислота. | FALSE | No |
| **Mode\_CIP\_Rinse\_RunReq** | Out | BOOL | – | =(ActiveMode\_UD==13 AND reqCIP\_Rinse)【48†】 | CIPMgr (in) | Запрос CIP-ополаскивание. | FALSE | No |
| **PAR\_UsePID** | Out | BOOL | – | MB\_ParMask.b0【13†】 | TempMgr (in)【60†】 | Флаг использования ПИД (передается в TempMgr). | *считывается* | Yes (via ParMask) |
| **PAR\_UseVFD** | Out | BOOL | – | MB\_ParMask.b1 | ActuatorMgr(in)【35†】, VFDMgr(in)【56†】 | Флаг использования ПЧ. | *считывается* | Yes |
| **PAR\_UsePortionButton** | Out | BOOL | – | MB\_ParMask.b2 | DosingMgr(in)【84†】 | Флаг использования внешней кнопки порции. | *счит.* | No |
| **PAR\_UseGunSensor** | Out | BOOL | – | MB\_ParMask.b3 | DosingMgr(in)【84†】 | Флаг контроля датчика “пистолет”. | *счит.* | No |
| **BLOCK\_AllOff** | Out | BOOL | – | calc: e.g. = NOT DI7\_EStop OR criticalAlarm | ActuatorMgr(in)【34†】 | Глобальная блокировка всех выходов. 1 = выключить всё (превалирует над ручным). Устанавливается при авариях (E-Stop=0 и др.). | FALSE | No |
| **BLOCK\_HeaterOff** | Out | BOOL | – | calc: e.g. = (DI2==0 OR DI3==0) | ActuatorMgr(in)【34†】, TempMgr(in?) | Блокировка нагрева. 1 = запретить включение DO6-DO8/AO1. Используется при аварии нагревателя или отсутствии воды. | FALSE | No |
| **STAT\_OutStateMask** | Out | UDINT | bits | calc bits from DO1-DO14 | Main (temp var) | Промежуточная маска состояний выходов. Затем копируется в MB\_OutStateMask. | 0 | No |
| **ALM\_Mask\_UD** | Out | UDINT | bits | calc bits from ALM\_\* | Main (temp var) | Промежуточная маска аварий (0–15 биты). Затем копируется в MB\_Alarms. | 0 | No |
| **MODE\_Current** | Out | INT | код режима | calc: e.g. 1..5 | HMI (MB\_SystemStatus) | Номер текущего режима (для индикации на HMI): 0 Idle, 1 Pasteur, 2 Manual, 3 CIP, 4 Dispense, 5 Recipe. Формируется на основании ActiveMode и ManualActive. | 0 | No |
| **MODE\_PasteurActive** | Out | BOOL | – | = (ActiveMode\_UD==1) | ActuatorMgr(in)【33†】, HMI | Флаг активности пастеризации. 1 все время, пока PasteurManager выполняет цикл. | FALSE | No |
| **MODE\_ManualActive** | Out | BOOL | – | = (Manual mode selected) | ActuatorMgr(in)【34†】, HMI | Флаг ручного режима. Включается при приоритете Manual. | FALSE | No |
| **MODE\_CIPActive** | Out | BOOL | – | = (ActiveMode\_UD>=10<=13) | ActuatorMgr(in)【33†】, HMI | Флаг CIP режима (любой). | FALSE | No |
| **MODE\_DispenseActive** | Out | BOOL | – | = (ActiveMode\_UD==2 or 3) | ActuatorMgr(in)【33†】, HMI | Флаг режима розлива (непрерывного или порционного). | FALSE | No |
| *(далее, возможно, отдельные сигналы для HMI, см. 1.3)* | – | – | – | – | – | – | – | – |

**Примечание:** Некоторые выходы Main фактически являются глобальными переменными Ram, которые не обязательно объявлены явно как Var\_Output, но записываются в Main. Например, ALM\_… флаги и MB\_\* регистры – Main формирует их. Выше включены лишь основные.

**Логика FB\_Main\_PR102 (алгоритм):** Выполняется циклически (примерно 10–100 мс цикл, предполагаем). Описываем блоками:

* **Считывание входов:** В начале цикла Main читает физические DI и AI. В OwenLogic глобальные переменные, привязанные к адресам, обновляются автоматически. Main использует их, например DI7\_EStop для блокировки, AI\_Temp… для вычислений. Также читаются регистры MB\_CmdMask1/2, MB\_ParMask, Heartbeats (через глобальные переменные, обновляемые драйвером Modbus).
* **Преобразование AI и диагностика датчиков:** Main вычисляет временные REAL tmpR для каждого датчика температуры【68†】. Проверяет на выход за допустимые пределы: если tmpR < TEMP\_MIN\_C or tmpR > TEMP\_MAX\_C – считает чтение невалидным (ALM\_TempProdBad=TRUE). В таких случаях поднимает соответствующий флаг аварии датчика:
* Если tmpR < -50.0°C, то ALM\_SensorProduct\_Short=TRUE (КЗ датчика продукта, бит1); если tmpR > 150.0°C – ALM\_SensorProduct\_Break=TRUE (обрыв, бит0). Аналогично для рубашки (бит3 и бит2).
* Если чтение валидно, обновляет переменную prodC\_ok := tmpR (последняя корректная T продукта)【68†】. Если не валидно – **оставляет предыдущий** prodC\_ok (тем самым при обрыве датчика TemperatureManager будет получать последнее значение, предотвращая резкие скачки). Это позволяет, например, при кратковременном сбое не сорвать цикл сразу, но через ALM уже идет сигнал оператору.
* Устанавливает PV\_TempProduct\_Raw := prodC\_ok【69†】; аналогично для Jacket: PV\_TempJacket\_Raw := jackC\_ok. Так TemperatureManager всегда получает “проверенные” значения (с фильтрацией обрыва).
* *Замечание:* Вместо удержания последнего, можно задать безопасное значение (например 0°C) при обрыве, но тогда TemperatureManager сам должен отключить нагрев по ALM. Решено: оставлять последнее корректное измерение, но сопровождать ALM флагами.
* Если обнаружен обрыв датчика в процессе пастеризации – PasteurManager увидит, что температура перестала изменяться и есть ALM – он инициирует аварийный останов (ALM\_PasteurAbort). Пока этого нет в коде, нужно добавить (см. 6.4).
* **Обработка команд HMI:** На основе считанных MB\_CmdMask1/2 формируется набор внутренних запросов:
* Например, если обнаружен фронт бита b0 Mask1 (Start Pasteur) – устанавливается reqPasteur := TRUE. Если пришел бит b1 (Stop Pasteur) – снимается reqPasteur (и, если пастеризация шла, устанавливается флаг для ALM\_PasteurAbort/норм стоп). Подобным образом для DispCont, DispDose, CIP.
* Эти внутренние флаги req *удерживаются Main до тех пор, пока оператор не снимет команду* *или*\* пока режим не станет активным и не будет выключен. Т.е. reqPasteur скорее всего сбрасывается либо при StopPasteur на HMI, либо автоматически по окончании цикла. Текущая реализация, судя по коду, может использовать latching: однажды получив команду, ActiveMode удерживает RunReq, а после окончания цикла PasteurManager сам сбрасывает Mode\_PasteurActive (Main видит и сбрасывает reqPasteur).
* После обработки, Main должен сбросить соответствующие биты в MB\_CmdMask (например, MB\_CmdMask1 := MB\_CmdMask1 & 0xFFFE для b0) – чтобы HMI не видела команду постоянно. **В текущем коде это не реализовано: необходимо добавить.**
* Отдельно обрабатывается CMD\_ResetAlarms (Mask1.b3): при его фронте Main сбрасывает все ALM\_\* флаги (кроме постоянно мониторящихся). ALM\_Mask, соответственно, обнуляется. Также можно моргнуть битом SystemStatus.b3 “Reset done”.
* **Выбор активного режима:** Main решает, какой режим получить управление, на основе req-флагов и приоритетов. Псевдокод:
* newMode = 0;  
  IF reqCIP\_Full THEN newMode=10;  
   ELSIF reqCIP\_Alk THEN newMode=11;  
   ELSIF reqCIP\_Acid THEN newMode=12;  
   ELSIF reqCIP\_Rinse THEN newMode=13;  
   ELSIF reqPasteur THEN newMode=1;  
   ELSIF reqDispCont THEN newMode=2;  
   ELSIF reqDispDose THEN newMode=3;  
   ELSE newMode=0;  
  END\_IF;  
  ActiveMode\_UD := newMode;
* Это соответствует тому, что CIP приоритетны (проверяются первыми)【49†】, затем Pasteur, затем Dispense. Если ни один не запрошен – режим Idle (0).
* Если ManualActive=TRUE (что устанавливается вне этого алгоритма, например, отдельным переключателем или по наличию нажатых ручных команд?), то все авто req должны игнорироваться. В коде FB\_Main явного Mode\_ManualActive входа нет – вероятно, Manual режим “возникает”, когда оператор начинает нажимать Man-биты. Однако, подход лучше: в HMI должен быть переключатель режимов Manual/Auto. Допустим, SystemStatus.b0 Manual может устанавливаться с панели. Тогда Main при b0=1 просто не рассматривает reqPasteur/CIP (ActiveMode остается 0, а ActuatorManager будет видеть Mode\_Manual\_ACTIVE=1 и работать по ManX). Такой функционал стоит добавить: например, если MB\_SystemStatus\_in.manual=1, то принудительно ManualActive. В реализацию ТЗ: ManualActive определяется либо внешним флагом, либо если хотя бы один ручной бит ManX=TRUE и автоцикл не активен – [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ].
* **Выставление RunReq:** На каждую систему-менеджер выводится свой RunReq сигнал (см. Var\_Output). Main устанавливает их согласно выбранному ActiveMode:
  + Если ActiveMode\_UD ==1 и reqPasteur=TRUE, то Mode\_Pasteur\_RunReq=TRUE (PasteurManager запускается)【47†】. Остальные RunReq = FALSE.
  + Если ActiveMode меняется на 0 (например, цикл окончен), Main сбрасывает все Mode\_\*\_RunReq.
  + Если активирован другой режим, соответственно. Для CIPManager важно: у него четыре входа RunReq (Full/Alk/Acid/Rinse). Main сделает один из них TRUE, остальные FALSE, в зависимости от выбранного подрежима CIP【48†】. CIPManager внутри знает, какой именно сценарий выполнять.
  + Для DosingManager: у него EN\_Continuous и EN\_Portion (две булевых)【84†】. Main подаст одну из них TRUE, другую FALSE (в коде: EN\_Continuous = (ActiveMode==2), EN\_Portion = (ActiveMode==3)).
* **Установка Active флагов:** После рассылки RunReq, Main обновляет глобальные флаги Mode\_XActive. В нашей схеме:
  + MODE\_PasteurActive = (PasteurManager.ReportRunning) – можно либо равнять ActiveMode==1, либо лучше пусть PasteurManager сам сообщает, что цикл действительно идет (например, как VarOutput). Но раз уж ActiveMode определяет, считаем: PasteurActive=TRUE, если ActiveMode=1 и PasteurManager не завершил. Здесь возможен рассинхрон: если пастеризация закончилась, reqPasteur может еще быть TRUE, но ActiveMode уже может быть сброшен на 0.
  + Чтобы точно знать активность, PasteurManager может выдавать сигнал CycleActive (например, пока не выполнены все стадии). В Main можно комбинировать: PasteurActive = Mode\_Pasteur\_RunReq OR (PasteurManager.ActiveStage >0).
  + Для простоты, будем считать: Mode\_PasteurActive = (ActiveMode\_UD==1) (то есть раз режим выбран). Аналогично CIPActive=(ActiveMode>=10), DispenseActive=(ActiveMode in 2,3). ManualActive – отдельный флаг. Эти флаги используются ActuatorManager, а также идут на HMI (SystemStatus).
* **Вызов менеджеров:** Main должен вызывать (или выполнять код) подчиненных FB в определенной последовательности, учитывая зависимые данные:
* **TemperatureManager** нужно вызывать **до ActuatorManager**, потому что TempMgr выдает команды нагрева, которые ActuatorMgr сразу примет на выходы. Но также TempMgr зависит от PasteurMgr (который определяет EN\_Heat и SP\_Target).
* **PasteurManager** выдает команды клапану охлаждения, мешалке и т.д., их Actuator использует. TempMgr ему нужен чтобы нагрев, но TempMgr вход EN\_Heat мы берем из PasteurMgr (через Main). То есть последовательность может быть:
  1. PasteurManager (на вход получает RunReq, SP\_ уставки и т.д., **но** TemperatureManager еще не обновлен – ему не нужно, PasteurMgr сам решает, разрешать ли нагрев). PasteurManager устанавливает у себя выход HeatingPowerReq (EN\_Heat), CoolingValve\_ON, CircPump\_ON и др.
  2. TemperatureManager (читает обновлённые PV\_Temp… Raw из Main, PAR\_UsePID, EN\_Heat от PasteurMgr – через глобальную переменную, либо Main напрямую соединяет – мы решим, что Main: TempMgr.EN\_Heat := PasteurMgr.HeatingPowerReq). TempMgr вычисляет новые DO\_HeaterX\_Cmd и AO\_HeaterPower.
  3. CIPManager (независимо: если CIPActive, ему RunReq дала Main. Он выставит CIP\_Valve\_ON, CIP\_Pump\_ON и т.п. на своих выходах).
  4. DosingManager (если DispenseActive, RunReq даны, он решает включать ли FillPump\_ON – на выходе).
  5. VFDManager (получает MixerRunCmd/MixerRevCmd – вероятно, Main формирует их: если PAR\_UseVFD=TRUE и, скажем, Pasteur\_Mixer\_ON=TRUE, то MixerRunCmd=TRUE. Но сделаем иначе: ActuatorManager сам в коде не использует VFD\_MixerRunReq (так как мы видели, не используется). Поэтому проще: Main: VFDManager.MixerRunCmd = (Mode\_PasteurActive AND Pasteur\_Mixer\_ON) OR (Mode\_CIPActive AND CIP\_Mixer\_ON) OR (Mode\_DosingActive??) OR (ManualActive AND Man3\_MixerFwd). Но ManualActive лучше напрямую -> Manual bits управляют DO. Скажем, Manual пуск мешалки через ПЧ – нет, ManualShould be possible. Поэтому, **важный момент**: при PAR\_UseVFD=TRUE, ручное управление мешалкой должно идти через VFDManager, а не через DO. То есть Man3/Man4 должны переводиться в MixerRunCmd/MixerRevCmd. В ActuatorManager сейчас: если ManualActive, он выставляет DO3/DO4 = Man3/Man4【106†】. Но при VFD=TRUE, DO3/DO4 физически не задействованы, однако Actuator все равно будет ставить AO2=0 (UseVFD) и DO3/DO4 мы можем считать “фиктивно” – их значения никуда не пойдут. Поэтому ManualRev/Forward можно параллельно использовать, отлавливая их в Main: если PAR\_UseVFD=TRUE и Mode\_ManualActive=TRUE, то MixerRunCmd = Man3\_MixerFwd, MixerRevCmd = Man4\_MixerRev. Это можно реализовать непосредственно в ActuatorManager (вход VFD\_MixerRunReq он имеет, но не использует – можно задействовать: подать туда ManualForward). Чтобы не запутаться: VFDManager лучше управлять из Main напрямую.
  6. ActuatorManager – **последним**. Он собирает все команды: Pasteur\_X\_ON, CIP\_X\_ON, Dosing\_FillPump\_ON, Temp\_DO\_HeaterX, Temp\_AO\_HeaterPower, Manual signals, блокировки, режимы. И выставляет физические DO/AO. Таким образом, к моменту его вызова все другие модули должны обновить свои выходы-глобалы.

В OwenLogic возможен параллельный запуск (все FB оцениваются “параллельно”, затем свод), но мы считаем последовательность вызовов: Pasteur -> CIP -> Dosing -> Temp -> VFD -> Actuator. Так обеспечивается, что Actuator возьмет актуальные значения текущего цикла.

* **Формирование выходных регистров:** После обновления ActuatorManager (который выставил Ram-переменные DO1..DO14 и AO1,AO2), Main создает STAT\_OutStateMask (например, обнулить, затем for i=1..14: if DOi then STAT\_OutStateMask |= (1 << (i-1))). Записывает MB\_OutStateMask := STAT\_OutStateMask (Modbus рег.517)【127†】. Аналогично, собирает ALM\_Mask\_UD: для бит0-15 проверяет соответствующие ALM\_\* флаги (см. список в 3.4), формирует 16-битное значение (в UDINT). Затем MB\_Alarms := ALM\_Mask\_UD (Modbus 512). Заполняет MB\_SystemStatus:
* b0 = MODE\_ManualActive,
* b1 = (MODE\_PasteurActive or MODE\_CIPActive or MODE\_DispenseActive) – т.е. хоть один авто,
* b2 = (ALM\_Mask\_UD != 0) – есть авария,
* b3 = (CMD\_ResetAlarms активен?) – можно мигнуть этим битом если пришла команда сброса. Записывает его в рег.519. Обновляет MB\_PLC\_Heartbeat (рег.544): просто инкрементирует на 1 (0→1→2… по мод 65535) – сигнал панели, что ПЛК жив. *Замечание:* Возможно, HMI будет делать сама – но лучше ПЛК тоже, как подтверждение связи.
* **Запись на выходы:** В OwenLogic физические DO/AO обновляются по привязанным переменным. После цикла ActuatorManager значения DO1..DO14 (Ram) отобразятся на выходах, AO1, AO2 – на аналоговых. Сразу после цикла Main можно считать, что команда на оборудование выдана.

Таким образом, FB\_Main\_PR102 координирует все модули, не содержa длительных собственных операций (кроме проверок датчиков). Он служит “клеем” между остальными частями.

**Взаимоблокировки и постусловия:** - Main отвечает за то, чтобы одновременно только один авто-режим имел RunReq=TRUE. Постусловие: сумма всех Mode\_*RunReq ≤ 1 (невозможно два одновременно TRUE). Проверка: код ActiveMode это гарантирует. Если ManualActive – тогда все ModeRunReq должны быть FALSE (это следует зафиксировать: при переключении в Manual, Main должен снять reqAuto). - Main устанавливает BLOCK\_AllOff немедленно при авариях – и держит его до устранения. Постусловие: при BLOCK\_AllOff=TRUE, все DO должны быть FALSE (ActuatorManager это выполняет). - Main должен сбрасывать аварии только по явной команде Reset и если причина исчезла. Постусловие: ALM* флаг не “залипает” после Reset, но может вновь взвестись, если аварийное условие ещё актуально (например, ResetAlarms нажали, а датчик всё ещё обрыв – ALM сразу вновь =1). Поэтому алгоритм ResetAlarms: сначала проверить, что исходная причина устранена (например, проверить DI/AI), и только потом сбросить. Впрочем, у нас ALM\_SensorXXX будут сброшены Reset’ом даже если датчик не исправен – но на следующем цикле Main опять их поставит, т.к. ALM\_TempBad останется. Это нормально.

### 2.2 FB\_ActuatorManager – менеджер исполнительных устройств

**Назначение:** Сосредоточивает управление всеми физическими выходами (digital outputs DO1..DO14, analog outputs AO1..AO2) в одном модуле. FB\_ActuatorManager принимает команды от других функциональных блоков (Pasteur, CIP, Dosing, Temperature) и от оператора (Manual), применяет правила приоритета и блокировок, и выставляет выходные переменные, соответствующие каждому DO/AO. Тем самым обеспечивается принцип **единственного писателя**: только ActuatorManager изменяет состояние DO/AO. Остальные модули лишь устанавливают “заявки” (флаги требуемых действий), которые ActuatorManager либо реализует, либо игнорирует (в зависимости от режима/блокировок).

Ключевые функции: - **Приоритет ручного управления:** Если активен ручной режим (Mode\_Manual\_ACTIVE=TRUE), ActuatorManager игнорирует автоматические команды и напрямую копирует значения ручных команд Man1..Man14 на соответствующие DO【41†】. Это реализовано набором конструкций IF … ELSIF … ELSE для каждой группы выходов. Ручные сигналы соответствуют 1:1 физическим выходам (Man1 -> DO1, Man2 -> DO2, … Man14 -> DO14)【31†】【32†】【33†】. Благодаря этому оператор в Manual может принудительно включить/выключить любой механизм независимо от автологики. - **Автоматическое управление (Auto):** Если ManualActive=FALSE, ActuatorManager ориентируется на флаги активных режимов (Mode\_Pasteur\_ACTIVE, Mode\_CIP\_ACTIVE, Mode\_Dosing\_ACTIVE, Mode\_Recipe\_ACTIVE) и включает выходы согласно заявкам соответствующих менеджеров: - Когда **PasteurActive OR RecipeActive = TRUE** (RecipeActive предполагается как вариант пастеризации по рецепту), ActuatorManager разрешает командам PasteurManager управлять рядом выходов: - DO1\_ValveCold = Pasteur\_CoolingValve\_ON (клапан охлаждения рубашки)【42†】, - DO2\_CircPump = Pasteur\_CircPump\_ON (насос циркуляции теплоносителя) – но только если CIP не активен (CIPActive имеет приоритет и выключает DO2)【43†】, - DO3\_MixerFwd = Pasteur\_Mixer\_ON (мешалка продукта)【107†】, - DO4\_MixerRev = FALSE (в режиме пастеризации реверс не используется)【108†】, - (DO5 не относится к пастеризации), - DO6/7/8 (нагреватели) – **НЕ** управляются PasteurManager напрямую! Вместо этого – TemperatureManager выдает Temp\_DO\_HeaterX\_Cmd, а PasteurManager лишь разрешает нагрев (EN\_Heat). Поэтому ActuatorManager присваивает DO6 = Temp\_DO\_Heater1\_Cmd, DO7 = Temp\_DO\_Heater2\_Cmd, DO8 = Temp\_DO\_Heater3\_Cmd при условии соответствующих блокировок (см. ниже). - DO9..DO14 – не относятся к пастеризации, поэтому PasteurActive -> они все = FALSE (если CIP не перехватил). - AO1 (аналог нагрева) = Temp\_AO\_HeaterPower (процент нагрева от TempMgr) – если нагрев не заблокирован. - AO2 (аналог мешалки) – PasteurActive: если PAR\_UseVFD=FALSE, а мешалка включена (DO3 true), устанавливает AO2 = 100% (10В)【89†】. Если PAR\_UseVFD=TRUE, AO2=0 всегда (мешалка управляется ПЧ). - Когда **CIPActive = TRUE**, ActuatorManager обслуживает CIPManager команды: - DO9\_ValveCIP\_Cold = CIP\_Valve\_Cold\_ON, - DO10\_ValveCIP\_Hot = CIP\_Valve\_Hot\_ON, - DO11\_ValveAlk = CIP\_Valve\_Alk\_ON, - DO12\_ValveAcid = CIP\_Valve\_Acid\_ON, - DO13\_PumpCIP = CIP\_Pump\_ON, - DO14\_ValveDrain = CIP\_DrainValve\_ON, - DO3\_MixerFwd = CIP\_Mixer\_ON (если CIP цикл требует мешалку)【107†】, - DO4\_MixerRev = FALSE (CIP не реверсирует мешалку), - Остальные DO (1-2,5-8) при CIPActive=TRUE выключаются (например, DO1\_ValveCold = FALSE, DO2\_CircPump = FALSE для исключения одновременного охлаждения/пастеризации во время CIP)【42†】【43†】. - AO1 – в CIP режиме нагрев продукта не используется (CIP подразумевает внешнюю горячую воду через DO10). Поэтому AO1 = 0 всегда при CIPActive. (Если в CIP задействован пар через рубашку, тогда CIPManager мог бы давать EN\_Heat – но такого не предусмотрено, CIP использует внешние среды.) - AO2 – при CIPActive, мешалка если включена (CIP\_Mixer\_ON) и PAR\_UseVFD=FALSE, ActuatorManager ставит AO2=100% (как и в Pasteur, т.к. CIP не управляет скоростью, просто включает). Если VFD, AO2=0. - Когда **DosingActive = TRUE** (Dispense mode): - DO5\_PumpDisp = Dosing\_FillPump\_ON【86†】 (насос розлива включается, если менеджер дозирования требует), - Остальные DO, если ни Pasteur, ни CIP не активны, остаются FALSE (если DispenseActive = TRUE, CIP/Pasteur точно FALSE, Manual=FALSE). Исключение: возможно, нужно мешалку крутить при Dispense? В коде Actuator мы видим, что если Mode\_Dosing\_ACTIVE, то для DO3/DO4 нет отдельной ветки – значит, они пойдут в ELSE (выключены)【108†】. Это означает: при розливе мешалка **не будет работать** автоматически. Если требуется мешать во время дозированного розлива (например, чтобы продукт не оседал), нужно добавить: ELSIF Mode\_Dosing\_ACTIVE THEN DO3\_MixerFwd := TRUE/FALSE?. Возможно, лучше использовать PAR\_StoreEnabled: если после пастеризации в режиме хранения (который реализован как часть DispenseActive?), то мешалка должна работать. **Решение:** предусмотреть опцию – [Альтернатива] Добавить параметр PAR\_MixDuringDispense – или использовать Manual to turn on if needed. Пока оставим: при DispenseActive мешалка по автомату не включается. - AO1 – в режиме розлива нагрев не нужен (цикл пастеризации завершен), но если PAR\_StoreEnabled=1, возможно TempManager продолжает поддерживать температуру (тогда Mode\_DispenseActive не наступит, вместо него PasteurActive остаётся? Проект неясен). Предположим, DispenseActive => нагрев выкл, AO1=0. - AO2 – если DispenseActive и мешалка вдруг нужна (что не реализовано), AO2 аналогично. - Когда **ни один авто-режим не активен (Idle)**: ActuatorManager находит все Mode\_...\_ACTIVE=FALSE и ManualActive=FALSE, тогда **все DO выключаются**. Код: в каждой ветке IF … ELSIF … есть заключительный ELSE, где DOx = FALSE【42†】【108†】. AO выходы тоже идут в ноль (в начале цикла ActuatorManager они уже обнуляются в “безопасных значениях” и не меняются, если ничего не включено).

* **Блокировки:** ActuatorManager имеет входные флаги BLOCK\_AllOff и BLOCK\_HeaterOff【34†】. Они применяются как безусловные отключения:
* Если BLOCK\_AllOff=TRUE, модуль *сразу после инициализации* не выполняет логику включения. В коде реализовано так:
* // Safe defaults:  
  DO1..DO14 := FALSE; AO1,AO2 := 0;  
  // ...  
  IF NOT BLOCK\_AllOff THEN  
   ... (вся остальная логика IF Mode\_xxx THEN ...)  
  END\_IF;
* То есть, при BLOCK\_AllOff=true, после сброса выходов в 0, модуль выйдет без изменения (все останется FALSE/0)【40†】. Таким образом, общий стоп/авария моментально гасит выходы.
* Если BLOCK\_HeaterOff=TRUE, внутри логики ActuatorManager, в секции нагревателей, имеется проверка: ST IF BLOCK\_HeaterOff THEN DO6:=FALSE; DO7:=FALSE; DO8:=FALSE; ELSE // нормальная логика нагрева (Manual/Pasteur) END\_IF;【118†】. Таким образом, даже если TemperatureManager просит нагрев (или Manual нажал), при активной блокировке Heater все три группы принудительно выключаются. Аналогично и AO1: мы видели, что AO1 присваивается Temp\_AO\_HeaterPower только **вне** BLOCK\_AllOff (и, фактически, вне блока BLOCK\_HeaterOff – хотя прямого IF нет, можно добавить: если BLOCK\_HeaterOff, то AO1:=0). Сейчас AO1 ставится = Temp\_AO\_HeaterPower всегда, но когда HeaterOff=TRUE, TempManager, скорее всего, уже сбросил Temp\_AO\_HeaterPower=0 сам (TempMgr увидит, что EN\_Heat=FALSE или Alarm). Но лучше явно: добавить в ActuatorMgr IF BLOCK\_HeaterOff THEN AO1:=0;.
* При блокировке AllOff, HeaterOff не требуются отдельно, но мы оставляем для случая, если нужно отключить нагрев, но остальные механизмы могут работать (например, нет воды в рубашке: нагрев Off, но мешалка, охлаждение можно оставить).
* **Другие блокировки:** PAR\_HeaterGroup2Enabled/Group3Enabled – реализованы внутри TemperatureManager, а ActuatorManager просто не получит команд на DO7/DO8 если они не разрешены (TempMgr не выставит DO\_Heater2/3\_Cmd=true). Тем не менее, Manual режим может включить их вручную невзирая на PAR – но Manual подразумевает, что оператор берет ответственность, поэтому это приемлемо.
* **Управление аналоговым выходом AO2 (мешалка):** Особенность – AO2 может выполнять роль задания скорости мешалки, если нет ПЧ. ActuatorManager решает: ST IF NOT PAR\_UseVFD THEN IF (DO3\_MixerFwd OR DO4\_MixerRev) THEN AO\_2 := 100.0; ELSE AO\_2 := 0.0; ELSE AO\_2 := 0.0; END\_IF;【89†】. То есть, если используется аналоговое управление (без ПЧ) и хоть одно направление мешалки активно, то на AO2 подается 100% (10 В), иначе 0. Тем самым, мешалка будет вращаться на максимальной скорости всегда. **Ограничение:** в данной реализации нельзя задать другую скорость (процент). Рекомендуется: задействовать параметр MB\_SetMixerFreq (0–1000) – например, AO2 = PAR\_MaxMixerSpeed \* (MB\_SetMixerFreq/1000) при включенной мешалке. PAR\_MaxMixerSpeed\_0\_1pct50 (рег.536) можно использовать как ограничение: например, оператор установил 800 (80%), AO2 = 8 В при включении. Пока этого нет – отметим как возможность улучшения.
* При PAR\_UseVFD=TRUE, AO2 всегда 0 (не используется). В этом случае VFDManager управляет скоростью через Modbus.
* Также, ActuatorManager не задействует DO3/DO4, если UseVFD=TRUE, для автоуправления – но **Manual** все равно ставит DO3/DO4. В итоге, если VFD=TRUE и ManualActive, можно получить: DO3=TRUE, AO2=0, ПЧ команд от ПЛК не получит (если не обрабатываем Man3). Решение: VFDManager должен тоже считывать Man3/Man4. Подробнее см. 2.7.
* **Управление аналоговым выходом AO1 (нагрев):** ActuatorManager напрямую передает значение от TemperatureManager: AO\_1 := Temp\_AO\_HeaterPower【120†】. При правильно настроенном масштабе Temp\_AO\_HeaterPower (0–1000) это выдает нужное напряжение 0–10В. Однако в текущей реализации TempManager выдает 0–100 (%). Требуется: либо в TempManager изменить шкалу на 0–1000, либо здесь домножить. Выбрано: TempManager работает в процентах 0–100 (удобно для ПИД), ActuatorManager домножает: AO1 := Temp\_AO\_HeaterPower \* 10. *Нужно внести исправление.* При BLOCK\_HeaterOff, ActuatorMgr должен установить AO1=0 (добавить).

**Таблица интерфейсов FB\_ActuatorManager:**

| Signal (FB\_ActuatorManager) | Dir | Type | Unit | Source / Consumers | Semantics and usage | Default | Retain |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Pasteur\_CircPump\_ON** | In | BOOL | – | PasteurManager.VarOutput【112†】 | Команда включить цирк. насос рубашки (охлажд. контур) в режиме пастеризации. В Actuator: влияет на DO2 (если PasteurActive). | FALSE | No |
| **Pasteur\_Mixer\_ON** | In | BOOL | – | PasteurManager.VarOutput | Команда включить мешалку продукта при пастеризации. Actuator: DO3 (FWD) включается, если PasteurActive. | FALSE | No |
| **Pasteur\_CoolingValve\_ON** | In | BOOL | – | PasteurManager.VarOutput | Команда открыть клапан охлаждения (хол. вода в рубашку) в пастеризации (стадия Cool). Actuator: DO1 включает при PasteurActive【42†】. | FALSE | No |
| **Pasteur\_HeatPowerReq** | In | REAL? | % | PasteurManager.VarOutput (Mismatch!) | ***Несогласовано***: предполагался вход требуемой мощности нагрева от Pasteur. Сейчас Pasteur выдает BOOL HeatingPowerReq. Будет исправлено на BOOL. Actuator в текущем коде его **не использует напрямую** (TempManager занимается нагревом). | 0.0 | No |
| **CIP\_Valve\_Cold\_ON** | In | BOOL | – | CIPManager.VarOutput | Команда открыть клапан холодной воды CIP (DO9). Actuator: DO9=ON при CIPActive. | FALSE | No |
| **CIP\_Valve\_Hot\_ON** | In | BOOL | – | CIPManager.VarOutput | Команда клапан горячей воды CIP (DO10). | FALSE | No |
| **CIP\_Valve\_Alk\_ON** | In | BOOL | – | CIPManager.VarOutput | Клапан подачи щелочи (DO11). | FALSE | No |
| **CIP\_Valve\_Acid\_ON** | In | BOOL | – | CIPManager.VarOutput | Клапан подачи кислоты (DO12). | FALSE | No |
| **CIP\_Pump\_ON** | In | BOOL | – | CIPManager.VarOutput | Команда включить насос CIP (DO13). | FALSE | No |
| **CIP\_DrainValve\_ON** | In | BOOL | – | CIPManager.VarOutput | Команда открыть клапан слива CIP (DO14). | FALSE | No |
| **CIP\_Mixer\_ON** | In | BOOL | – | CIPManager.VarOutput | Команда включить мешалку во время CIP (опционально, задается для этапов CIP). Actuator: DO3=ON при CIPActive【107†】. | FALSE | No |
| **Dosing\_FillPump\_ON** | In | BOOL | – | DosingManager.VarOutput【95†】 | Команда включить насос розлива (DO5) во время режима Dispense. Actuator: DO5 = ON при DosingActive【86†】. | FALSE | No |
| **Temp\_DO\_Heater1\_Cmd** | In | BOOL | – | TemperatureManager.VarOutput【62†】 | Команда включить нагреватель группы1 (DO6). Actuator: если не заблокирован нагрев, DO6 = TRUE при поступлении этой команды (когда TempManager решает, что надо греть). | FALSE | No |
| **Temp\_DO\_Heater2\_Cmd** | In | BOOL | – | TemperatureManager.VarOutput | Команда на группу2 (DO7). Только если PAR\_HeaterGroup2Enabled=TRUE, TempMgr может дать TRUE. ActuatorManager просто передает на DO7 (при отсутствии блок). | FALSE | No |
| **Temp\_DO\_Heater3\_Cmd** | In | BOOL | – | TemperatureManager.VarOutput | Команда на группу3 (DO8). | FALSE | No |
| **Temp\_AO\_HeaterPower** | In | REAL | % | TemperatureManager.VarOutput | Аналоговая команда мощности нагрева (0–100%). Actuator: преобразует в AO1 (0–10V). **После фиксации**: TempMgr будет выдавать 0–1000, Actuator прям на AO1. | 0.0 | No |
| **VFD\_MixerRunReq** | In | BOOL | – | VFDManager (или Main) – не используется сейчас | Резерв на будущее: запрос “мешалка вперед” для Actuator, если хотим DO3 использовать для запуска ПЧ. Пока ActuatorManager его не обрабатывает. | FALSE | No |
| **VFD\_MixerRevReq** | In | BOOL | – | – | Резерв: запрос “мешалка реверс”. Аналогично не используется в Actuator. | FALSE | No |
| **Man1\_ValveCold** (CmdMask2.b0) | In | BOOL | – | HMI (Modbus команд 2)【54†】 | Ручная команда клапан охлаждения рубашки. При Mode\_Manual\_ACTIVE Actuator: DO1 = Man1. | FALSE | No |
| **Man2\_CircPump** (b1) | In | BOOL | – | HMI | Ручная команда цирк. насос рубашки. Actuator: DO2 = Man2 (ManualActive). | FALSE | No |
| **Man3\_MixerFwd** (b2) | In | BOOL | – | HMI | Ручной “Мешалка Вперед”. Actuator: DO3 = Man3 (ManualActive). При VFD=TRUE физически DO3 не задействован, но Actuator все равно ставит. (В будущем – можно ловить в VFDManager). | FALSE | No |
| **Man4\_MixerRev** (b3) | In | BOOL | – | HMI | Ручной “Мешалка Назад”. Actuator: DO4 = Man4. (Обычно реверж не используется, но предусмотрено на случай мешалки реверсивной.) | FALSE | No |
| **Man5\_PumpDisp** (b4) | In | BOOL | – | HMI | Ручное управление насосом розлива. Actuator: DO5 = Man5 (ManualActive). | FALSE | No |
| **Man6\_Heater1** (Mask1? b?) | In | BOOL | – | HMI (команды Manual?) | Ручное управление нагревателем 1. Actuator: DO6 = Man6, но **внимание:** Manual может насильно включить нагрев даже вне TemperatureManager. Это допускается, но оператор должен соблюдать осторожность. | FALSE | No |
| **Man7\_Heater2** | In | BOOL | – | HMI | Ручной нагрев 2 (DO7). | FALSE | No |
| **Man8\_Heater3** | In | BOOL | – | HMI | Ручной нагрев 3 (DO8). | FALSE | No |
| **Man9\_ValveCIP\_Cold** | In | BOOL | – | HMI | Ручной клапан ХВС CIP (DO9). | FALSE | No |
| **Man10\_ValveCIP\_Hot** | In | BOOL | – | HMI | Ручной клапан ГВС CIP (DO10). | FALSE | No |
| **Man11\_ValveAlk** | In | BOOL | – | HMI | Ручной клапан щёлочи CIP (DO11). | FALSE | No |
| **Man12\_ValveAcid** | In | BOOL | – | HMI | Ручной клапан кислоты CIP (DO12). | FALSE | No |
| **Man13\_PumpCIP** | In | BOOL | – | HMI | Ручной насос CIP (DO13). | FALSE | No |
| **Man14\_ValveDrain** | In | BOOL | – | HMI | Ручной клапан слив (DO14) / зуммер. | FALSE | No |
| **Mode\_Pasteur\_ACTIVE** | In | BOOL | – | FB\_Main (Mode flag)【42†】 | Флаг: режим пастеризации сейчас активен. Используется для выбора, чьи команды выполнять. | FALSE | No |
| **Mode\_CIP\_ACTIVE** | In | BOOL | – | FB\_Main (flag)【43†】 | Флаг: CIP активен. | FALSE | No |
| **Mode\_Dosing\_ACTIVE** | In | BOOL | – | FB\_Main (flag) | Флаг: режим розлива активен. | FALSE | No |
| **Mode\_Manual\_ACTIVE** | In | BOOL | – | FB\_Main (flag)【41†】 | Флаг: ручной режим активен. Определяет ветку логики Manual. | FALSE | No |
| **Mode\_Recipe\_ACTIVE** | In | BOOL | – | (не реализован?) | Флаг: режим “рецептурный” (если есть сложная пастеризация). В коде присутствует, но может не использоваться. | FALSE | No |
| **BLOCK\_AllOff** | In | BOOL | – | FB\_Main【34†】 | Блокировка всех выходов. | FALSE | No |
| **BLOCK\_HeaterOff** | In | BOOL | – | FB\_Main【34†】 | Блокировка нагрева. | FALSE | No |
| **PAR\_UseVFD** | In | BOOL | – | FB\_Main (ParMask b1)【35†】 | Параметр: использовать ли ПЧ для мешалки. Определяет поведение AO2 и DO3/DO4. | FALSE | Yes (via retain) |

**Выходы ActuatorManager (Var\_Output):** – это, по сути, сами глобальные переменные DO/AO, привязанные к физическим каналам. Имена выходов **совпадают с именами глобальных сигналов** для простоты (как указано комментарием “имена строго как в FB\_Main”):

| Signal | Dir | Type | Unit | Consumers (Physical) | Semantics | Default | Retain |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **DO1\_ValveCold** | Out | BOOL | – | Физ. выход DO1, MB\_OutStateMask bit0 | Клапан охлаждения рубашки (подача холодной воды). Включается при PasteurActive (охлаждение) или вручную. | FALSE | No |
| **DO2\_CircPump** | Out | BOOL | – | DO2, Mask bit1 | Циркуляционный насос контура пастеризатора (прогон теплоносителя). Работает при PasteurActive (при нагреве/охлаждении) или вручную. | FALSE | No |
| **DO3\_MixerFwd** | Out | BOOL | – | DO3, Mask bit2 | Мешалка – направление “Вперёд”. При PasteurActive или CIPActive – включается, если требовалось мешание. При ManualActive – по команде Man3. Если UseVFD=TRUE, этот сигнал может идти на “сухой” выход, т.к. управление через ПЧ. | FALSE | No |
| **DO4\_MixerRev** | Out | BOOL | – | DO4, Mask bit3 | Мешалка – направление “Назад”. (Опционально, если мешалка реверсируемая). Используется только вручную (Man4), автоматически всегда FALSE (CIP/Pasteur не реверсируют). | FALSE | No |
| **DO5\_PumpDisp** | Out | BOOL | – | DO5, Mask bit4 | Насос розлива продукта. Включается в режиме Dispense (непрерывно или пока льется порция) или вручную. | FALSE | No |
| **DO6\_Heater1** | Out | BOOL | – | DO6, Mask bit5 | Нагреватель (группа 1) или клапан пара (если PAR\_UseSteam=1, то DO6 мог бы быть клапаном пара). Управляется TemperatureManager (если EN\_Heat=TRUE и нужна мощность) либо вручную. Отключается при BLOCK\_HeaterOff. | FALSE | No |
| **DO7\_Heater2** | Out | BOOL | – | DO7, Mask bit6 | Нагреватель группа 2. Работает, если TempManager решил задействовать доп. группу (обычно при большой разнице температуры или при %PID > порога). Учитывается PAR\_HeaterGroup2Enabled. Можно включить вручную. | FALSE | No |
| **DO8\_Heater3** | Out | BOOL | – | DO8, Mask bit7 | Нагреватель группа 3. Аналогично группе 2 (PAR\_HeaterGroup3Enabled). | FALSE | No |
| **DO9\_ValveCIP\_Cold** | Out | BOOL | – | DO9, Mask bit8 | Клапан холодной воды CIP. Включается CIPManager во время тех циклов, где нужна холодная вода. Или вручную (Man9). | FALSE | No |
| **DO10\_ValveCIP\_Hot** | Out | BOOL | – | DO10, Mask bit9 | Клапан горячей воды CIP. Включается CIPManager (этапы с горячей водой) или вручную (Man10). | FALSE | No |
| **DO11\_ValveAlk** | Out | BOOL | – | DO11, Mask bit10 | Клапан подачи щелочи CIP. Включается CIPManager (циклы щелочи) или вручную. | FALSE | No |
| **DO12\_ValveAcid** | Out | BOOL | – | DO12, Mask bit11 | Клапан подачи кислоты CIP. | FALSE | No |
| **DO13\_PumpCIP** | Out | BOOL | – | DO13, Mask bit12 | Насос CIP. Включается CIPManager на этапах циркуляции или вручную. | FALSE | No |
| **DO14\_ValveDrain** | Out | BOOL | – | DO14, Mask bit13 | Клапан слива CIP (и по совместительству, возможно, зуммер – если CIP не используется, этот выход задействуют под зуммер сигнализации). Включается CIPManager на этапах слива или вручную (Man14). | FALSE | No |
| **AO\_1** (Analog1) | Out | REAL | 0–1000 (0–100%) | AO Channel 1 (0–10V) | Аналоговый сигнал управления нагревом. Если PAR\_UseSteam=0 (электр.нагреватели), этот выход может не использоваться (вместо него – дискретные DO6-8). Если PAR\_UseSteam=1, этот сигнал идет на клапан подачи пара (пропорционально). В любом случае AO1 отображает текущую %мощность нагрева (0–100%). Формируется TemperatureManager (PID) и выдается Actuatorом. | 0.0 | No |
| **AO\_2** (Analog2) | Out | REAL | 0–1000 (0–100%) | AO Channel 2 (0–10V) | Аналоговый сигнал управления скоростью мешалки. Используется, если PAR\_UseVFD=FALSE (аналоговый привод): 100% = максимальная скорость (ограничена PAR\_MaxMixerSpeed). Если PAR\_UseVFD=TRUE, AO2 всегда =0 (не используется). | 0.0 | No |

*Примечания:* Retain для выходов – нет, т.к. при перезагрузке все механизмы должны стартовать выключенными (исключение – если нужен “память состояний”, но здесь не нужно).

**Алгоритмы внутри ActuatorManager:** Частично уже объяснены. Ниже суммируем с акцентом на предотвращение конфликтов: - **Защита от одновременного включения прямого и реверса мешалки:** Код должен гарантировать, что DO3 и DO4 никогда не TRUE одновременно. В реализации: Manual режим – зависит от оператора (может ли он нажать Man3 и Man4 одновременно? Интерфейс скорее не позволит, но если да, Actuator просто выставит оба, что физически должно быть блокировано межзамком проводки или логикой ПЧ). Auto режим – всегда либо FWD либо ничего (CIP/Pasteur никогда не активируют Rev). Так что нормально. Если нужно – можно добавить: IF DO3 && DO4 then DO4:=FALSE (например) – но, полагаем, не требуется. - **Единственный писатель DO/AO:** Все присваивания DOx и AOx происходят *только* внутри ActuatorManager (блок “safe defaults” + ветки IF). Проверкой исходников подтверждено: нигде вне ActuatorManager DO не присваиваются【40†】. Это инвариант, который сохраняется. - **Реакция на Manual override:** Когда ManualActive включается, ActuatorManager мгновенно переключает выходы по Man-битам, *не дожидаясь отключения автосигналов другими модулями*. Например, шла пастеризация, DO2 был TRUE от Pasteur\_CircPump\_ON, но как только ManualActive=TRUE, Actuator сделает DO2 := Man2 (скорее всего Man2=FALSE если оператор его не удерживает) и т.д. То есть Manual может внезапно оборвать некоторые процессы (охлаждение и т.п.) – что допустимо, ведь оператор сознательно вмешивается. - **Возврат из Manual в Auto:** Как только ManualActive сброшен (и ManX все 0), ActuatorMgr пойдет в ветки Mode\_xxx. Если при этом какой-то авто-режим был на паузе, его флаги Mode\_ACTIVE могут тут же воздействовать. Например, оператор вошел в Manual на середине CIP – CIPManager по RunReq все еще думает, что цикл идет (не получал Stop). После выхода из Manual, ActuatorMgr снова увидит Mode\_CIP\_ACTIVE=TRUE и начнет открывать клапаны CIP как ни в чем не бывало. Это нежелательно – поэтому в разделе 1.2 указывалось, что вход в Manual должен сопровождаться сбросом авто-режимов. Вводим требование: **При переключении Manual→Auto оператор должен повторно запустить нужный цикл**, а не продолжать с середины. В реализации: добавить в ManualActivate: if going Manual, set all reqPasteur/CIP… = FALSE, и при выходе ManualActive=FALSE ActiveMode останется 0 (Idle). - **Автоматическое отключение неиспользуемых систем:** Благодаря разделению по режимам, когда, например, CIPActive=TRUE, ActuatorMgr гарантированно выключает всё, относящееся к пастеризации и розливу (DO1,2,5…8) вне зависимости от их команд. И наоборот, PasteurActive отключает CIP выходы. Это предотвращает конфликты (например, чтобы одновременно не подалась вода в рубашку и CIP). - **Задержки и др.:** ActuatorManager сам по себе не использует таймеры задержки – он мгновенно переключает выходы по входным флагам. Если нужны задержки (например, не включать насос CIP без открытия клапанов), это решается внутри CIPManager или аппаратно (через обратные клапаны).

В конце цикла ActuatorManager все Var\_Output отражают требуемые состояния выходов. ПЛК применяет их к реальному оборудованию.

### 2.3 FB\_TemperatureManager – регулирование температуры

**Назначение:** Модуль управления нагревом продукта в пастеризаторе. Получает текущие температуры продукта и рубашки, целевую температуру пастеризации, решает – включать нагрев или нет (в режимах нагрева/выдержки), с какой мощностью. Поддерживает два алгоритма: ПИД-регулятор (для плавного контроля) или простое двухпозиционное регулирование с гистерезисом (для систем, где ПИД не требуется). Также распределяет нагрузку между тремя группами нагревателей (включает/выключает группы 1,2,3), реализует ограничение темпа нагрева (через уставку ΔT между рубашкой и продуктом), минимальные времена включения/выключения для ТЭНов (anti-short-cycling), а также защищает от перегрева (датчики, ∆T) и формирует аварийные сигналы, связанные с температурой.

**Интерфейс:** TemperatureManager получает на вход уже *валидированные* температуры (REAL), а также параметры и разрешающие сигналы:

| Signal (FB\_TemperatureManager) | Dir | Type | Unit | Source (from Main/Pasteur) | Semantics | Default | Retain |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **PV\_TempProduct\_Raw** | In | REAL | °C | FB\_Main (конверсия AI)【60†】 | Текущая температура продукта (сырое значение без сглаживания), обновляется каждый цикл. Если датчик не в порядке, значение “последнее корректное”. | 0.0 | No |
| **PV\_TempJacket\_Raw** | In | REAL | °C | FB\_Main【60†】 | Температура рубашки (теплоносителя) текущая. Используется для ∆T защиты и логики охлаждения/нагрева. | 0.0 | No |
| **PAR\_UsePID** | In | BOOL | – | FB\_Main (ParMask b0)【60†】 | Выбор алгоритма: TRUE = ПИД-регулирование нагрева, FALSE = простое включение/выключение по гистерезису. | TRUE | Yes (retain ParMask) |
| **EN\_Heat** | In | BOOL | – | PasteurManager (или Main)【59†】 | “Разрешение нагрева” – сигнал от пастеризации: 1 = мы на стадии, где нужно греть (Heat или Hold); 0 = нагрев не требуется (например, этап охлаждения, пауза или авария). Если FALSE, TemperatureManager отключает все нагреватели независимо от ПИД. | FALSE | No |
| **SP\_TempTarget** | In | REAL | °C | PasteurManager.SP\_HeatTemp【76†】 | Целевая температура продукта (обычно 72–95°C) для достижения и поддержания. Задается оператором (HMI) через Main. Используется как уставка ПИД или порог в гистерезисе. | 0.0 | No |
| **Seconds\_UD** | In | UDINT | sec tick | FB\_Main (global second)【61†】 | Секундный тик для внутренних таймеров. Не изменяется внутри TempMgr, но используется для расчета интервалов. | – | No |
| **PAR\_PWM\_Period\_s** | In | UDINT | s | (retain param, вручную задать)【61†】 | Период ШИМ для управления ТЭНами при ПИД. Рекомендация: 1–2с при SSR, 10–30с при контакторах. Оптимально, чтобы нечасто щелкало, но достаточно часто для регулирования. TempMgr использует для формирования импульсов для DO6-8 при ПИД (чтобы аналоговую мощность превратить в дискретные вкл/выкл). | 15 (пример) | Yes |
| **PAR\_G2\_StartPct** | In | REAL | % | (retain param)【61†】 | Порог мощности для включения группы нагрева 2. Например, 60%: если требуемая мощность >60%, TempMgr включает DO7, иначе держит его выкл. Помогает задействовать доп. ТЭН при больших нагрузках. | 60.0 | Yes |
| **PAR\_G3\_StartPct** | In | REAL | % | (retain param)【61†】 | Порог включения группы 3. Например, 85%. | 85.0 | Yes |
| **PAR\_HeaterGroup2Enabled** | In | BOOL | – | FB\_Main (Par retain)【61†】 | Разрешение на использование группы 2. Если FALSE, TempMgr никогда не включает DO7 независимо от расчета. Позволяет отключить ТЭН2, если не установлен. | TRUE | Yes |
| **PAR\_HeaterGroup3Enabled** | In | BOOL | – | FB\_Main (Par retain)【61†】 | Разрешение группы 3. | TRUE | Yes |
| **PAR\_UseSteam** | In | BOOL | – | FB\_Main (Ram, Params) | Признак “используется пар вместо электричества”. Влияет: если TRUE, TempMgr может рассчитывать AO\_HeaterPower, но не активировать DO6-8 (кроме, возможно, DO6 как клапан пара). Этот флаг может обрабатываться либо здесь, либо в Actuator (в данном проекте Actuator не различает, он просто AO1 всегда выдает). В TempMgr можно: если UseSteam=TRUE, всегда держать DO\_Heater1/2/3\_Cmd=FALSE и выдавать только аналоговую мощность. | FALSE | Yes |
| **Hyst\_HeatRequest** | Out? | BOOL | – | (в коде есть Ram var) | Флаг “запрос нагрева по гистерезису”. Возможно, когда PAR\_UsePID=FALSE, TempMgr устанавливает Hyst\_HeatRequest=TRUE, если нужно нагреть. Первоначально эта переменная была в Main, но сейчас мы делаем внутри TempMgr. Ее можно использовать как “нагрев идет” – например, PasteurManager может видеть. | FALSE | No |

**Выходы TemperatureManager:**

| Signal | Dir | Type | Unit | Consumers | Description | Default | Retain |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **DO\_Heater1\_Cmd** | Out | BOOL | – | ActuatorMgr.In【62†】 | Команда включить нагреватель 1. TempMgr выставляет TRUE, если нужно подавать мощность (по алгоритму). Если PAR\_UseSteam=TRUE, может использоваться как команда клапану пара (DO6). | FALSE | No |
| **DO\_Heater2\_Cmd** | Out | BOOL | – | ActuatorMgr.In | Команда нагрев 2. Включается, если требуется доп.мощность > PAR\_G2\_StartPct *и* разрешено PAR\_HeaterGroup2Enabled. В случае PWM, может переключаться (общий цикл PWM). | FALSE | No |
| **DO\_Heater3\_Cmd** | Out | BOOL | – | ActuatorMgr.In | Команда нагрев 3 (аналогично группе 2, порог PAR\_G3\_StartPct). | FALSE | No |
| **AO\_HeaterPower** | Out | REAL | % | ActuatorMgr.In【62†】 | Аналоговый выходной сигнал мощности (0–100%). При PAR\_UsePID=TRUE – это расчет ПИД, ограниченный 0–100%. При PAR\_UsePID=FALSE – возможно используется как индикатор или усредненная скважность. ActuatorMgr передает на AO1 (0–1000). | 0.0 | No |
| **PV\_TempProduct** | Out | REAL | °C | PasteurMgr (мониторинг), HMI | Отфильтрованное (сглаженное) значение температуры продукта. TempMgr может сглаживать raw сигнал (например, экспоненциально). Выдается для стабильности отображения и вычислений. PasteurManager может смотреть, достигли ли SP (лучше по фильтрованному). | = Raw initially | No |
| **PV\_TempJacket** | Out | REAL | °C | PasteurMgr, HMI | Отфильтрованная температура рубашки. Используется для контроля ∆T. | = Raw | No |

**Примечание:** Retain не нужен – после перезапуска ПЛК нагрев выключен, PV сбрасываются.

**Внутренние переменные и алгоритмы:** Опишем, как TempManager работает: - **Фильтрация температуры:** Возможна реализация цифрового фильтра (например, скользящее среднее или экспоненциальное). Судя по названию “Soft” и “Raw/PV”, предполагается простое сглаживание. Можно использовать коэффициент α (например, 0.1) для экспоненциального: PV\_TempProduct := PV\_TempProduct + α\*(PV\_TempProduct\_Raw - PV\_TempProduct). В начальном состоянии PV=Raw. Фильтр убирает шум датчиков, особенно полезно для ПИД. Параметры фильтра – статически заданы (α=0.2…0.3). В коде не явлено явно, но документируем: *Настроено экспоненциальное сглаживание температур с коэффициентом ~0.2 (обновление за ~5 циклов на 63%).* - **Переключение режимов (PID/гистерезис):** - Если PAR\_UsePID=TRUE: TempManager использует ПИД-регулятор для поддержания PV\_TempProduct = SP\_TempTarget. Необходимо задать коэффициенты (Kp, Ki, Kd). В проекте они могли задаваться где-то (например, PID\_Kp, PID\_Ti, PID\_Td), но нет явных. Пока можно считать, что внутри TempMgr либо фиксированные настройки, либо их нет (ALM\_PIDBadParams = TRUE если не настроено). - ПИД выдает непрерывную величину Output % (0–100). - Этот Output – **общая мощность**, требуемая от нагревательной системы. TempManager должен разложить ее на использование групп: *Принцип работы с группами:* Например, у нас три группы ТЭНов по ~1/3 мощности каждая. Тогда: - Если Output <= ~33%, задействуется только группа1, с PWM подачей соответствующего процента. - Если Output > PAR\_G2\_StartPct (скажем 60%), то постоянно включаем группу1 и 2, и оставшуюся долю модулируем на них (либо на одной из групп). Но лучше: - При >60%: включаем группу1 и 2 на 100% (итого ~66% мощности) и далее начиная с 66% до ~85% используем PWM на группу1 (т.к. 2 группы уже 66%, нужно довполнить до нужного %). - Если >85%: включаем группу1,2,3 на 100% (≈100% мощности). Если Output <100 но >85, можно группу3 PWM. *Проще:* - Сравниваем Output с порогами: - if Out > G3\_StartPct (85): все три DO=TRUE (100% мощность). - else if Out > G2\_StartPct (60): DO1, DO2 = TRUE; DO3 (grp3) = FALSE; а группу1/2 можно PWM совместно для требуемого процента (но т.к. они полностью on, overshoot). - На самом деле, overshoot: при Out=70% ( >60 <85) – 2 группы дадут ~66%, нужно еще ~4% от третьей: можно включать третью на 4/33 = 12% времени (PWM). - Можно PWM не группой1, а последней требуемой группой (в данном случае grp3 PWM 20%). - Если Out <= 60: DO2,DO3 = FALSE; DO1 PWM на Out%. - Если 60 < Out <= 85: DO1, DO2 = TRUE (на 100%); рассчитать excess = Out - 66%, и PWM DO3 на (excess/33%)\*100% цикла. - Если 85 < Out < 100: DO1,DO2,DO3 = TRUE (100%), но по сути Out max = 100, все включено. Этот алгоритм обеспечивает грубую каскадную дискретизацию. - Реализация PWM: TempManager знает Seconds\_UD, можно завести цикл PWMPeriod. Например, PWM\_Period=15с. Можно хранить текущее время начала цикла pwm\_start (UDINT). Каждый цикл TempMgr вычисляет t = Seconds\_UD - pwm\_start (секунд прошло). Если t > PAR\_PWM\_Period\_s, то pwm\_start = Seconds (новый цикл). Доля времени, на которую надо включить одну из групп = (PWM\_percent \* Period). Например, Out=50%, period=10с: Heater1 должен быть включен 5с из 10, выключен 5с. Можно проще: - Завести суммирующий счетчик “pwm\_accum” (повышается на Output \* dt). - Каждый шаг dt=1с, Output% \* period /100. - Ладно, точность секунды может быть мала для pwm, лучше 1с шаг – 10с period – 10% резолюция. - Оставим: реализуется на уровне кода, не вдаемся очень – главное, TempMgr выдает DOx\_Cmd=TRUE или FALSE каждый цикл, поддерживая среднее время. - ПИД защита: если параметры ПИД нулевые или некорректные (Kp=0), TempMgr может выставить ALM\_PIDBadParams (бит7) и перейти к гистерезису. - Если PAR\_UsePID=FALSE: TempManager работает как двухпозиционный регулятор с гистерезисом. - Вероятно, используется переменная Hyst\_HeatRequest: - Пока PV\_Product < SP\_Target - Δ, включить нагрев (HeatRequest=TRUE), пока PV\_Product не превысит SP\_Target + Δ, после чего выключить (HeatRequest=FALSE) и не включать, пока не опустится ниже SP\_Target - Δ. - Δ можно задать либо жестко (~1°C) либо использовать SP\_SmoothDiff (т.к. он 0.1°C *some?). SP\_SmoothDiff задан 0.1°C? Нет, в таблице SP\_SmoothDiff\_x10=526, по умолчанию likely 0 (в Pasterizator\_Стандартные\_Переменные.csv SP\_SmoothDiff: “диффер. выравнивания, °C”). Possibly SP\_SmoothDiff = 2.0°C by default? Actually, in CSV snippet【8†】 it shows “SP\_SmoothDiff\_x10: 0,1°C”, likely meaning e.g. 10 = 1.0°C. - Но PasteurManager использует SP\_SmoothDiff для охлаждения гистерезиса【79†】, знач 2-3°C. - Можно использовать половину от него для нагрева гистерезиса, либо вводим фикс. - Когда HeatRequest=TRUE, TempMgr включает соответствующие DO Heater (всегда все группы? Либо тоже каскадно, но проще: гистерезис, возможно, только группа1? Или все?). Обычно, при двухпозиционном, можно включать все группы если крупный аппарат (быстрее нагреть), или тоже поэтапно: при небольшом перегреве одну группу отключить. - Упростим: Если UsePID=FALSE, TempMgr контролирует только DO\_Heater1\_Cmd (основной нагрев). DO2/DO3 можно просто не использовать (или оператор сам вручную включает при большой загрузке). Либо можно включать все группы одновременно.* Предположение:\* В данном проекте гистерезис, возможно, был задуман как “грубый режим”, где либо все ТЭНы On, либо Off по потребности. Т.е. HeatRequest=TRUE => DO6,7,8 = TRUE (если разрешены) – вся мощность. That might overshoot, but hence the Hyst was mainly for non-critical uses. - Hyst\_HeatRequest (Ram bool) – можно выставлять = TempMgr.Мы хотим нагрев. Main или PasteurManager в предыдущей версии могли читать Hyst\_HeatRequest и управлять нагревателями. Но сейчас, раз TempMgr сам командует DO, глобал Hyst\_HeatRequest можно просто для индикации. - ALM по перегреву: e.g. ∆T. Если PV\_TempJacket - PV\_TempProduct > SP\_SmoothDiff (например 5°C), можно временно отключать нагрев даже если продукт не достиг целевого – чтобы не перегреть рубашку (избегать пригорания). - В TempMgr можно: if (PV\_Jacket > PV\_Product + SP\_SmoothDiff) then форсировать отключение Heater (имитировать, что достигли). PasteurManager, скорее, тоже наблюдает ∆T (в коде Pasteur упоминается “защита от переохлаждения рубашки”【79†】, т.е. при охлаждении). - TempMgr может делать “защита от перегрева рубашки”: если Jacket слишком горячая относительно продукта, и EN\_Heat=TRUE, то можно отложить включение нагрева до снижения ∆T. (Наблюдалось на практике: если налить слишком горячую воду при начале нагрева, продукт может пригореть). Implementation: if PV\_Jacket > PV\_Product + X, then не включать нагрев (даже если EN\_Heat=TRUE) пока ∆T не уменьшится. SP\_SmoothDiff как раз, кажется, = 2.0°C – можно использовать. - Минимальные времена On/Off: желательно, чтобы ТЭНы не щелкали слишком часто. Например, MinOn=30с, MinOff=30с (для контакторов). Implement: - Если только что включили нагрев (DO\_Heater1\_Cmd TRUE), то держать минимум MinOn before allow off by Hyst (даже если температура перевалила порог). - Если выключили, не включать обратно раньше MinOff. - Для простоты, можно time stamp last switch and ensure. If needed – implement.

**Логика включения нагревателей:** - TemperatureManager **непосредственно** управляет DO\_Heater1/2/3\_Cmd. ActuatorManager лишь транслирует их на физические DO (с учетом блокировок). - Таким образом, TempMgr решает, сколько групп задействовать: - Если PAR\_HeaterGroup2Enabled=FALSE – никогда не устанавливает DO\_Heater2\_Cmd TRUE. - Если PAR\_HeaterGroup3Enabled=FALSE – аналогично для 3. - Если PAR\_UseSteam=TRUE – предполагается, что DO6 управляет клапаном пара (группа1), а групп 2 и 3 нет. Тогда TempMgr должен: - выдать AO\_HeaterPower как основную регулировку (0–100%), - и, возможно, DO6\_Cmd как сигнал “подать пар” (в On/Off режиме, например, DO6\_Cmd=EN\_Heat – чтобы паровой клапан открыт только на фазах нагрева/выдержки, но не при охлаждении). Actually, если пар – обычно аналоговый клапан, do we need DO6? - В hardware, DO6 might open main steam valve (fully open when heating allowed) and AO1 modulates control valve. If that’s the design, then: - EN\_Heat (Pasteur’s heating stage) = DO6\_Cmd (open steam line), - AO1 (via AO\_HeaterPower) controls regulating valve to maintain temp. - This approach uses both DO6 and AO1. If so, PAR\_UseSteam triggers this logic: - When EN\_Heat=TRUE, DO\_Heater1\_Cmd=TRUE always (open steam), - and AO\_HeaterPower regulated by PID (control valve). - If EN\_Heat=FALSE, DO\_Heater1\_Cmd=FALSE (close steam). - DO\_Heater2/3\_Cmd permanently FALSE (no electric). - В ТЗ: зафиксируем: *При PAR\_UseSteam=TRUE, TempMgr включает DO6 как “паровой клапан” (простой открыто/закрыто) на стадиях, разрешенных EN\_Heat, а AO\_HeaterPower формирует задание на регулирующий орган (0–100% поток пара).*

* **Аварии и ограничения:**
* *Датчики:* Main уже формирует ALM\_SensorX. TempMgr может дополнительно реагировать:
  + Если ALM\_SensorProduct\_Break or Short = TRUE, то TempMgr должен немедленно сбросить все команды нагрева (DO\_Cmd=FALSE, AO=0) и, возможно, блокироваться (например, запомнить состояние Fault). PasteurManager уже должен будет оборвать цикл.
  + Также можно ALM\_PIDBadParams: если ПИД неправильно настроен, TempMgr мог бы установить ALM\_PIDBadParams.
  + ALM\_HeaterFault (DI2=0) – Main устанавливает BLOCK\_HeaterOff, TempMgr видит, что нагрев отключен. Может, TempMgr даже не узнает, но Actuator off.
  + ALM\_NoWaterJacket (DI3=0) – Main BLOCK\_HeaterOff, TempMgr EN\_Heat=TRUE? It won't matter, DO remain off. Perhaps not needed inside TempMgr anything else.
* *∆T защита:* Обозначим допустимую разницу T между рубашкой и продуктом, выше которой нагрев приостанавливают, как Max\_DeltaT (можно использовать SP\_SmoothDiff или отдельный параметр). Например, Max\_DeltaT=20°C – если рубашка > продукт на 20°, нагрев off.
  + Actually, SP\_SmoothDiff in code is small (like 2°C?), which likely meant something else. Usually ∆T safety could be ~15°C to avoid burning.
  + Need confirmation. Possibly misnamed: "Diff. for smooth heating" suggests during ramp heating, keep jacket not more than X deg above product. This likely is the case: if product is far from target, they might want to avoid overheating jacket beyond target + X.
  + Implementation: If PV\_TempJacket > min(PV\_TempProduct + Max\_DeltaT, SP\_TempTarget + Max\_DeltaT2), then hold off heaters.
  + Because if product at 20°C and target 80°C, raising jacket to 80°C+20=100°C might risk burn, so maybe allow 20 deg difference. If beyond, wait for product to catch up (like dynamic soak).
  + Without further info, say: we implement basic ∆T cutoff: if PV\_Jacket > PV\_Product + 15°C, then do not turn on heaters even if EN\_Heat=TRUE, until difference <=15.
  + Mark [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ].
* *Min On/Off times:* If contactors controlling TЭН, to avoid chatter. We incorporate:
  + Use last\_on\_time, last\_off\_time internal. If we want fine control might skip due to PWM, but if SSR, no need. Maybe ignore if SSR (not given).
  + Could param decide: if PWM\_period >5 sec, contactor, apply min times (like 30).
  + [Опция] implement for safety: if DO was turned on, ensure at least 10s on; if turned off, at least 10s off. But if overshoot, too bad. Possibly skip as complicates.
  + If implement: use Seconds\_UD to enforce.
* *Охлаждение vs нагрев:* TempMgr doesn’t handle cooling directly (клапан охлаждения управляется PasteurMgr). Но TempMgr может помочь:
  + Если EN\_Heat=FALSE (например, идет охлаждение), TempMgr просто держит все DO\_Cmd=FALSE.
  + PasteurMgr при охлаждении может все же хотеть, чтобы если рубашка стала слишком холодной относительно продукта (например, при store mode, чтобы не переохладить продукт), приостановить охлаждение. Actually, PasteurMgr does that【79†】. So not in TempMgr, but PastMgr ensures not overshoot below target by closing valve. For heating, see above (TempMgr does similar with ∆T).

**Алгоритм TemperatureManager (по шагам):**

1. **Обновление фильтров:** PV\_TempProduct := Filter(PV\_TempProduct, PV\_TempProduct\_Raw), PV\_TempJacket := Filter(PV\_TempJacket, PV\_TempJacket\_Raw). Фильтр может быть простейшим: PV = Raw (в первой версии, действительно, Main подавал Raw, гистерезис делался снаружи; сейчас решено внутрь поместить).
2. **Проверка EN\_Heat:** Если EN\_Heat = FALSE, то **выключить нагреватели**:
3. DO\_Heater1\_Cmd = DO\_Heater2\_Cmd = DO\_Heater3\_Cmd = FALSE,
4. AO\_HeaterPower = 0.0,
5. Можно сразу выходить (не считать PID). Потому что если режим не требует нагрева (например, охлаждение, пауза, CIP), то ничего не делаем.
6. Исключение: PAR\_UseSteam=TRUE – возможно, хотим AO\_HeaterPower все равно считать? Но EN\_Heat=FALSE → значит и пар не нужен. AO1=0 (закрыть клапан).
7. Hyst\_HeatRequest = FALSE.
8. **Если EN\_Heat = TRUE:** (то есть пастеризация в фазе нагрев или поддержания):
9. **Режим PID:** если PAR\_UsePID=TRUE:
   * Проверить ∆T защиту: если PV\_TempJacket > PV\_TempProduct + Max\_DeltaT (скажем, 15°C), *или* PV\_TempJacket > SP\_TempTarget + someMargin (например, 5°C), то **блокируем нагрев** временно:
   * Можно реализовать как: не увеличивать PID Output или принудительно ограничить Output так, чтобы Jacket не перегревалась. Но проще: при слишком большом ∆T, ставим все DO=false и Output=0 (как будто EN\_Heat=false) до снижения ∆T. Сигнализируем об этом? (ALM\_NoWater maybe, or a new ALM if water present but difference too high, not exactly an alarm, more protective measure).
   * Пока можно не делать ALM, просто логика.
   * Вычислить ошибку: error = SP\_TempTarget - PV\_TempProduct.
   * ПИД: Out = Kp\*error + … (интеграл, дифф). Настройки?
   * Предположим, настроено.
   * Ограничить Out 0..100%.
   * Если Out > 100, saturate 100; <0, saturate 0 (но <0 маловероятно, т.к. error negative – продукт перегрелся – хотя возможно overshoot после hold, тогда Out=0, а PasteurMgr может вообще снять EN\_Heat).
   * ALM\_PIDBadParams: если Kp,Ki,Kd – какие-то заведомо плохие (например, Kp=0 or unstable), TempMgr может вывести Alarm (bit7) и fallback. [В отсутствии конкретики, опустим].
   * **PWM или аналог:**
   * Если PAR\_UseSteam=TRUE: у нас AO\_HeaterPower = Out (процентов пара). DO\_Heater1\_Cmd=TRUE (стимул открыть клапан) – но AO=Out% уже дает позицию клапана?
     + Подумать: если AO1 – регуляторный клапан, то DO6 может быть лишним. Пар возможно можно просто аналогом регулировать без бинарного клапана. Но часто ставят соленоид пар (вкл/выкл) + регулирующий клапан, или только регулирующий.
     + P&ID likely has one control valve (analog) and a solenoid safety maybe.
     + If we interpret DO6 as a main steam valve (for fail-safe shut off), it should open during heating/hold. So do: DO\_Heater1\_Cmd = TRUE always if EN\_Heat.
     + AO\_HeaterPower = Out. That covers analog.
     + DO2/DO3 remain not used.
   * Если PAR\_UseSteam=FALSE (электронагрев):
     + AO\_HeaterPower – можно использовать как индикатор, но само регулирование идет через дискреты.
     + Рекомендуется: AO\_HeaterPower = Out, но Actuator anyway uses DOs. The plan:
     + Рапределить Out на группы:
       - if Out==0: все DO false (easy).
       - if 0 < Out <= G2\_StartPct: только DO\_Heater1\_Cmd = PWM по Out%, DO2,DO3=FALSE.
       - if Out > G2\_StartPct: DO1, DO2 = ON, DO3 = off initially. Compute effective power if 2 on = ~66%.
       - rem = Out - 66% (if Out=70, rem=4%).
       - calculate PWM\_duty for DO3 = rem/(33%) \* 100% = ~12% duty.
       - So implement a PWM on DO3: e.g., if (t mod PWM\_Period) < (PWM\_Period \* PWM\_duty3) then DO3\_Cmd=TRUE else FALSE.
       - if Out > G3\_StartPct: DO1, DO2, DO3 = ON (100%).
       - If Out e.g. 90%, all on, overshoot 90 vs 100? Actually, if 3 on yield 100% heating capacity, Out=90 means you want slightly less than full, which can't be precisely achieved with 3 discrete full on. Could PWM one group off occasionally to yield ~90. But then it's similar to previous step where Out just >85. Actually, better:
       - if Out > G3\_StartPct (85), DO1, DO2 ON, DO3 PWM as above to achieve Out%. e.g. Out=90 -> after 66 from 2, rem=24% out of 33 => ~73% duty on third group.
       - if Out ~100, then effectively all ON.
       - if Out exactly 100 (or >99): all ON constantly.
     + This approach ensures roughly proportionate heat output with 3 steps.
     + DO\_Heater1\_Cmd we can use as continuous base (likely always ON when any heat needed, since group1 presumably always present).
     + Actually, need to consider if Out small, but >0, maybe not saturate group1 fully but PWM it as well:
     + Maybe simpler: always use PWM on one group to fine tune, while others either fully on or off:
       - If Out < G2\_StartPct: only group1 PWM at Out% (others off).
       - If G2\_StartPct <= Out < G3\_StartPct: group1 & 2 on 100%, group3 PWM = (Out-66)/(33) %.
       - If Out >= G3\_StartPct: group1 & 2 on, group3 PWM = (Out-66)/33% (for 85% this yields ~58% duty? Actually at 85, rem=19 -> 58% duty, at 100, rem=34 -> ~103% cap, saturate at 100% duty i.e. full ON).
       - Cap group3 PWM at 100%. So effectively:
       - Out in [85,100] -> group3 on full (since beyond 85, also saturate maybe to avoid undervaluing top range).
       - But then output between 85 and 100 will all yield 3 groups full (maybe minor overhead, but not catastrophic).
     + Conclusion:
     + DO1\_Cmd: ON if Out > 0 (with possible PWM if Out small? Eh, let's just go with ON always if any heat needed, because turning off group1 if Out small would require group2 or 3 which we don't want).
     + DO2\_Cmd: ON if Out > G2\_StartPct, else off.
     + DO3\_Cmd: ON if Out > G3\_StartPct, else PWM if Out between 66 and 85, else off if below 66. Actually:
       - If Out <=66: DO3 off.
       - If 66<Out<85: group3 PWM as per above.
       - If Out >=85: DO3 ON.
     + DO1 always on if any heat, perhaps it might overshoot for Out, but group1 presumably lowest power group or needed minimal for distribution? Perhaps group1 power ~1/3 total, so leaving it on for any Out ensures at least ~33% power even if Out wanted 10%. That overshoots significantly (target 10 but giving 33).
     + Not good for small Out.
     + Better: allow group1 to PWM too for low outputs:
       - Actually, simpler: handle groups sequentially:
       - Compute required number of groups = ceil(Out / 33%)
       - If one group needed (Out<=33): group1 PWM at Out/33 \* 100% (i.e. at 10% Out, group1 on ~30% of time).
       - If two groups needed (33<Out<=66): group1 ON, group2 ON maybe (should we PWM or ON?), group3 off. Actually with 2 groups, each is ~33% capacity, combined 66%. If Out=50%, perhaps one group on fully, second group on ~50%. But turning second group half-time gives smaller incremental step but might cause thermal cycling. Some do alternate which group toggles to even wear.
       - Possibly better:
       - For Out 33-66: group1 on, group2 PWM such that total = Out%. e.g. Out=50 -> total needed 50% out of 66 possible with 2 groups, so group2 duty ~ (50-33)/33 = 51%.
       - If Out=60 -> group2 duty ~ (60-33)/33 = 82%.
       - If Out just above 33, group2 duty ~ few %.
       - If Out>66 and <=100: group1 & 2 on, group3 PWM for remainder as earlier.
       - This ensures group1 always fully on once Out>33, group2 gradually engages from 0 to 100% duty as Out goes 33 to 66, group3 engages beyond 66 up to 100 similarly.
       - At Out=34, group1 on, group2 ~3% duty, group3 off => output ~33 + 1 =34%. Good.
       - At Out=66, group1 on, group2 ~100% duty (output ~66), group3 off.
       - At Out=67, group1 & 2 on, group3 ~3% duty (output ~66 +1=67).
       - At Out=100, all on output ~100.
       - This method uses PWM on group2 and 3 sequentially.
     + Let's adopt that approach in description:
10. **Switching algorithm final:**
    * Let Out\_percent = required heating % (0-100).
    * If Out <= 0: DO1,DO2,DO3 = FALSE.
    * Else:
    * If Out <= PAR\_G2\_StartPct (say 60):
      + DO1\_Cmd = TRUE (base group always on to ensure stable heating).
      + Compute duty2 = (Out - 33) / 33 if Out > 33 else 0 (because if Out under 33, one group could have been partly on; but since we forced DO1 fully on, we overshoot for Out<33. Actually, to correct overshoot for Out<33, we should PWM DO1 for those cases).
      + Correction: if Out < 33:
      + DO1\_Cmd should not be fully on, should PWM as well. So break it further:
      + If Out <= 33:
        - DO1 PWM at Out/33 \*100% duty,
        - DO2,DO3 off.
      + If Out > 33 and <= 66:
        - DO1 on,
        - DO2 PWM at (Out-33)/33 \*100%,
        - DO3 off.
      + If Out > 66 and < 100:
        - DO1 on,
        - DO2 on,
        - DO3 PWM at (Out-66)/34 \*100%.
      + If Out = 100:
        - DO1,2,3 on.
      + That covers all ranges. We use fixed 33% because presumably each group ~1/3 power. It's simpler to assume equal power distribution.
    * Implementation PWM:
    * We have Seconds\_UD for coarse time. For better resolution, one might want ms tick. But PР102 likely has system tick.
    * Possibly ignore complexity of code writing. We can just state logic.
11. **Outputs assignment:** Based on above logic, set DO\_Heater1/2/3\_Cmd and AO\_HeaterPower.
12. If PAR\_UsePID=TRUE and UseSteam=FALSE: we do PWM logic for DO's and AO\_HeaterPower = Out (for info).
13. If UseSteam=TRUE: DO1=EN\_Heat (open solenoid if heat needed), DO2,3=FALSE (no electric), AO\_HeaterPower = Out (to modulate steam).
14. If PAR\_UsePID=FALSE (hysteresis mode):
    * If Hyst\_HeatRequest not set and PV < SP\_Target - HysDelta: set Hyst\_HeatRequest=TRUE.
    * If Hyst\_HeatRequest set and PV >= SP\_Target + HysDelta: set Hyst\_HeatRequest=FALSE.
    * Then if Hyst\_HeatRequest=TRUE (and EN\_Heat=TRUE): DO1,2,3 = ON (if respective PAR\_Enable true). If only one group physically installed but others disabled, fine. Possibly do not modulate analog at all in this mode (AO\_HeaterPower just 0 or 100?).
    * If Hyst\_HeatRequest=FALSE: all DO off.
    * AO\_HeaterPower maybe can reflect "100 or 0", or better indicate proportion of time it was on (like if you want gauge on HMI).
    * But if using all groups at once, no analog needed except maybe to show heating element usage. We can output 100 when on, 0 when off.
15. Actually, hysteresis with all groups on might overshoot heavily especially in small volumes. Possibly they intended hysteresis to only use one group (lowest power) to reduce overshoot for small difference? Not sure. But they gave 3 group enabling as param, maybe expecting some always disabled if not needed.

**Примечание по Hyst\_HeatRequest:** В стандартных переменных Hyst\_HeatRequest объявлен как Ram bool, non-retain. Вероятно, раньше (когда TempMgr не было) Main на основании PV/SP вычислял HeatRequest гистерезис и напрямую управлял DO6-8 (в старой версии). Сейчас, раз TempMgr взял на себя – можно использовать Hyst\_HeatRequest как: TempMgr sets it = TRUE когда нагрев просит, FALSE когда нет. Можно вывести как VarOutput (добавить) или просто напрямую писать глобал (так как он Ram). - Он может пригодиться PasteurManager: например, если Hyst\_HeatRequest=FALSE (нагрев достиг точки), PasteurManager может стартовать таймер выдержки (Hold начинается с момента, когда продукт достиг целевой температуры). В отсутствии PID, момент достижения SP можно определить, когда HeatRequest впервые сбросился (плюс гистерезис). - Если PID, threshold crossing detection could be when PV >= SP maybe.

So, for consistency: - Provide Hyst\_HeatRequest as VarOutput if needed or global. - We'll mention it (already in interface possible out).

**Выработка аварий TemperatureManager:** - **ALM\_PIDBadParams (бит7):** Если PID mode, но параметры некорректны (Kp<=0, Ti… etc) – TempMgr может поднять этот флаг. Standard var ALM\_PIDBadParams exists. - **ALM\_NoWater (Ram Alarm)**: There is ALM\_NoWater (Boolean, "НО датчик потока/наличия воды" – interesting, seems bit beyond 15?). Actually ALM\_NoWater not in initial 16 mask (they had 0-15 bits), maybe separate or beyond: - It's listed after ALM\_Mask in CSV: ALM\_NoWater;Boolean;…;Ram;Alarms (no bit comment). Possibly an independent alarm outside mask? Or could correspond to CIP flow sensor (if no water in CIP loop). - Could be if DI6\_ProductFlow remains 0 while CIP running (lack flow -> pump cavitation). - If CIP uses DI6, CIPMgr should raise ALM\_NoWater. - Might not be Temp-related, skip.

* Temperature overshoot:
* If PV\_Product > SP\_Target + someMargin (like 2°C) – not necessarily alarm, perhaps just overshoot, could alarm if risk quality. Usually, short overshoot isn't critical if under pasteurization.
* But maybe if overshoot beyond allowed, ALM\_PasteurAbort triggers or ALM\_HeaterFault (if runaway).
* We'll skip.

**Взаимодействие с PasteurManager:** - PasteurManager определяет EN\_Heat (когда начать нагрев и когда закончить). TempMgr полностью подчиняется EN\_Heat. Например, если пастеризация завершилась (Hold done, Cool stage), PasteurMgr ставит EN\_Heat=FALSE – TempMgr немедленно отключает нагреватели. - PasteurManager также, вероятно, контролирует процесс достижения температуры: - Идея: PasteurMgr может считать, что этап нагрева закончился, когда продукт достиг SP (в динамике лучше дождаться, пока PV\_Product >= SP\_Target и, например, держится >=X секунд). - Когда это условие выполнено, PasteurMgr перейдет к этапу Hold (выдержка) – но при этом EN\_Heat останется TRUE, потому что нужно поддерживать температуру. - Отличие: на этапе Heat PasteurMgr мог требовать, чтобы SP достигли. На этапе Hold, EN\_Heat=TRUE продолжает, TempMgr через PID уже минимально держит. - Когда истек HoldTime, PasteurMgr ставит EN\_Heat=FALSE (нагрев больше не нужен, переходим к охлаждению). - Summation: - Stage Heat: EN\_Heat=TRUE; PasteurMgr ждет PV >= SP (monitors TempMgr.PV\_TempProduct). - Possibly use Hyst\_HeatRequest if non-PID: when it goes FALSE first time, we consider temp reached (since turned off heaters means overshoot above SP). - Stage Hold: EN\_Heat=TRUE; PasteurMgr запускает таймер выдержки. TempMgr поддерживает SP. - Stage Cooling: EN\_Heat=FALSE; TempMgr off, PasteurMgr opens ValveCold etc.

**Таблица работы TemperatureManager (пример):**

| Условие (пример) | DO\_Heater1\_Cmd | DO\_Heater2\_Cmd | DO\_Heater3\_Cmd | AO\_HeaterPower | Комментарий |
| --- | --- | --- | --- | --- | --- |
| EN\_Heat=0 (охлаждение/останов) | 0 | 0 | 0 | 0% | Нагрев выкл. |
| EN\_Heat=1, PAR\_UsePID=FALSE, PV<<SP | 1 | 1 (если Group2Enabled) | 1 (если Group3Enabled) | – | Все нагреватели включены (нагрев до достижения SP). |
| EN\_Heat=1, PAR\_UsePID=FALSE, PV>=SP+Δ | 0 | 0 | 0 | – | Превышена уставка + гистерезис: нагрев выкл. (в режиме hold). |
| EN\_Heat=1, PAR\_UsePID=TRUE, Output=50% (2 группы) | 1 | PWM ~51% | 0 | ~50% | Группа1 постоянно, группа2 включается ~51% времени, группа3 не используется. |
| EN\_Heat=1, UsePID, Output=80% (3 группы) | 1 | 1 | PWM ~42% | 80% | Группы1,2 постоянны, группа3 ~42% duty (общая мощность ~80%). |
| EN\_Heat=1, UsePID, UseSteam=TRUE, Out=40% | 1 (паровой клапан) | (no elect) | (no elect) | 40% | Паровой клапан открыт, подача пара регулируется аналоговым клапаном 40%. |
| EN\_Heat=1, ∆T слишком велика | 0 | 0 | 0 | 0% | Приостановка нагрева (ждет снижения ∆T). |

*(цифры условные, PWM duty illust.)*

### 2.4 FB\_PasteurManager – цикл пастеризации

**Назначение:** Автоматизирует процесс пастеризации продукта. Отвечает за переходы между фазами нагрева, выдержки и охлаждения, а также (опционально) фазой хранения продукта. Управляет вспомогательными механизмами: циркуляционным насосом рубашки, мешалкой продукта, клапаном охлаждения. Генерирует разрешение на нагрев (для TemperatureManager) в нужные периоды. Отслеживает таймер выдержки и ограничение времени на разогрев. Реагирует на аварийные ситуации (преждевременная остановка, пропадание нагрева, E-Stop). Обеспечивает, чтобы режим пастеризации выполнялся полностью до конца либо прерывался безопасно.

**Режим работы:** PasteurManager включается при RunReq=TRUE (Mode\_Pasteur\_RunReq от Main). До этого находится в пассивном состоянии (все выходы = FALSE). При активном RunReq, PasteurManager проходит последовательно стадии: 1. **PRE-HEAT (подготовка)** – (возможно, отсутствует как отдельная, сразу Heat). 2. **HEAT (нагрев до заданной температуры):** - Включается нагрев (EN\_Heat=TRUE) – PasteurMgr устанавливает соответствующий флаг【59†】. - Включается циркуляционный насос рубашки (CircPump\_ON=TRUE) – для обеспечить циркуляцию теплоносителя. - Мешалка продукта (Mixer\_ON=TRUE) – обычно на время нагрева и выдержки мешалка должна работать для равномерности прогрева. - Клапан охлаждения (CoolingValve\_ON=FALSE) – закрыт, чтобы рубашка могла нагреваться. - PasteurMgr ждет, пока продукт достигнет уставки SP\_HeatTemp. Он может отслеживать TempManager.PV\_TempProduct (отфильтрованную) или, если ПИД, можно дождаться PV\_TempProduct >= SP\_HeatTemp. - **Опция SkipHeat:** Если пришла команда SkipHeat (например, продукт уже горячий и нужно сразу выдержку) – PasteurManager может пропустить эту фазу: он проверяет CMD\_SkipHeatStage (Mask1.b2). При наличии, сразу переходит к фазе Hold, при этом EN\_Heat возможно даже не включается (или сразу выключается), чтобы не греть. - **Лимит времени нагрева:** Если разогрев затянулся более SP\_HeatTime минут (например, пастеризатор не смог достичь температуры), PasteurMgr должен зафиксировать ситуацию: - Установить STAT\_HeatTimeout=TRUE (Ram флаг) – “истек лимит времени нагрева”. - Оставить цикл в ожидании действия оператора: возможно, при HeatTimeout стоит остановить нагрев и поставить паузу с ожиданием – в ОИКП такой сценарий: когда время нагрева вышло, оператору предлагается либо продолжить (skip hold) или abort. - Пока оставим: STAT\_HeatTimeout=TRUE может блокировать продолжение цикла (например, PasteurManager не перейдет к Hold, пока оператор не даст CMD\_SkipHeatStage или Reset). - В ТЗ: указать, что при HeatTimeout требуется ручное вмешательство (подтверждение продолжения без достижения температуры). - При достижении температуры (или по SkipHeat/оператору) – переход к следующей стадии. 3. **HOLD (выдержка при заданной температуре):** - Начало выдержки фиксируется: PasteurMgr запускает таймер на SP\_HoldTime секунд. - EN\_Heat остается TRUE, чтобы TempManager поддерживал температуру (не дать продукту остыть). - CircPump\_ON = TRUE (продолжает циркуляция), Mixer\_ON = TRUE (мешалка продолжает). - Клапан охлаждения закрыт (CoolingValve\_ON=FALSE) – поддержание T идет с помощью нагрева (TempManager). - Пока идет таймер, PasteurMgr следит: если вдруг температура упадет ниже SP (например, резко тепло унеслось – маловероятно при регулируемом нагреве) – TempManager сам включится, все нормально. - По истечении HoldTime PasteurMgr регистрирует успешное пастеризование. Можно установить некий флаг “стерилизация достигнута” (не требуется, просто логика). - Переход к следующей стадии. 4. **COOL (охлаждение продукта до температуры хранения/слива):** - PasteurMgr ставит EN\_Heat = FALSE (нагрев более не нужен). - Отключает нагреватели через TempManager, оставляет мешалку и циркуляционный насос, чтобы активно охлаждать. - Открывает клапан охлаждения (CoolingValve\_ON = TRUE) – холодная вода подается в рубашку. - Циркуляционный насос рубашки (CircPump\_ON) – **вопрос**: нужно ли при охлаждении гонять теплоноситель? Если рубашка подключена к холодной воде напрямую (скорее всего проточно) – тогда циркуляционный насос контур может быть не нужен (его можно отключить). - Но, возможно, система: рубашка имеет одну магистраль подачи холодной, другая – горячей. Если холодная подается напрямую (типа душирующая), то цирк. насос, который гонял нагретый теплоноситель, можно отключить, чтобы не мешал. - Однако, может и нет отдельного – может, этот же насос гоняет холодную воду тоже. - Нужен анализ: DO2\_CircPump = “прокачка теплоносителя по контуру”. Если холодная вода подается из водопровода, то этот насос, возможно, только для нагрева (в замкнутом контуре?). - Предположим: пастеризатор с рубашкой имеет электротэны в рубашке + внешний контур не нужен, или если пар – тоже не нужен насос, но у нас DO2. - Вероятнее: DO2 – циркуляция теплоносителя (например, вода или гликоль) через внешний теплообменник (в случае электротэнов?), или, если нагрев паром, то насос может качать охлаждающую воду (в случае отсутствия давления водопровода). - *Решение:* при охлаждении **оставляем DO2\_CircPump=TRUE**, чтобы продолжать движение холодной воды. Либо, если DO2 – именно “насос нагрева”, можно отключить. - Program snippet【43†】 shows: ELSIF Mode\_CIP\_ACTIVE THEN DO2\_CircPump := FALSE, ELSIF Mode\_PasteurActive OR Mode\_DosingActive THEN DO2\_CircPump := Pasteur\_CircPump\_ON OR Dosing??. Possibly they allowed Dosing to reuse circ pump for store? But Dosing doesn’t manage circ pump. - The code suggests DO2 off in CIP, on in Pasteur or Dosing. So for PasteurActive entire time they likely keep DO2 on. - Keep approach: For all PasteurActive (Heat,Hold,Cool), DO2=TRUE (since Pasteur\_CircPump\_ON presumably always set true from start to end). - *Hysteresis/охлаждение защита:* Когда продукт достигнет SP\_CoolTemp (целевой охлаждения, например 30°C) – нужно остановить охлаждение. PasteurMgr не знает этого SP? Actually yes, we have SP\_CoolTemp (HMI reg.568). - После пастеризации часто продукт либо разливают горячим, либо охлаждают до темп розлива (say 20°C). - Если PAR\_StoreEnabled=FALSE, может быть, SP\_CoolTemp=0 (не используется) – тогда, может, и не охлаждаем? unlikely, pasteurization usually requires rapid cooling to stop cooking. - Считаем: SP\_CoolTemp != 0 => нужно охладить до этой температуры. PasteurMgr: - контролирует PV\_TempProduct, и когда PV <= SP\_CoolTemp, считает охлаждение завершенным. - Затем: закрывает клапан охлаждения (CoolingValve\_ON=FALSE), может оставить мешалку (если далее хранение) или выключить. - Если PAR\_StoreEnabled=TRUE, то после достижения CoolTemp, PasteurMgr не останавливает цикл, а переходит в стадию **STORE**. - Если StoreDisabled, то цикл завершается. - **Защита от переохлаждения рубашки:** В процессе охлаждения, если рубашка становится слишком холодной относительно SP\_CoolTemp, можно временно закрывать клапан, чтобы не переохладить продукт ниже цели. Это реализовано: - В коде Pasteur есть: IF PV\_TempJacket <= (SP\_CoolTemp - SP\_SmoothDiff) THEN CoolingValve\_ON := FALSE ELSE TRUE【79†】. То есть: когда рубашка остыла ниже (целевое - дифф), клапан закрывается (дать продукту догреться), когда рубашка нагреется выше целевой - дифф, клапан открывается. Это гистерезис охлаждения ~ SP\_SmoothDiff. - Например, SP\_CoolTemp=30°C, SP\_SmoothDiff=2°C: - Пока рубашка >28°C, держим клапан открыт, охлаждаем. Если рубашка остыла до 28 или ниже, закрываем клапан (даже если продукт >30, чтобы не переморозить). Когда рубашка снова нагреется (от продукта) выше 28, откроем снова. - Это предотвращает “перехлаждение” продуктовой емкости. - PasteurMgr таким образом имплементирует *гистерезис охлаждения*. - TemperatureManager в охлаждении не участвует (EN\_Heat=FALSE), так что это чисто Pasteur logic. - Когда продукт достиг SP\_CoolTemp, или даже если SP\_CoolTemp не задан (0) – возможно, PastMgr просто ждет некоторое минимальное время. - Дальше: 5. **STORE (хранение продукта):** *Опциональная стадия*, включаемая если PAR\_StoreEnabled=TRUE. После охлаждения до нужной температуры, продукт может поддерживаться при этой температуре (чтобы не потерять пастеризованность) до разлива. - Как реализовать: - PasteurMgr при StoreEnabled=1 после Cool не останавливает цикл, а: - оставляет циркуляцию (CircPump\_ON=TRUE?), мешалку (Mixer\_ON=TRUE?) – вероятно да, чтобы продукт не остывал/не расслаивался. - EN\_Heat = FALSE или TRUE? Странно: “режим хранения” может предполагать подогрев поддержания – но мы уже охладили до CoolTemp (например, 4°C в случае молока?). Если хранение, возможно, на ~60°C (термостатирование). - Возможен иной сценарий: StoreEnabled=1: после пастеризации, вместо охлаждения до низкой, продукт держат, скажем, на 60°C для горячего розлива. Но тогда SP\_CoolTemp = 60 and they maintain it. Or “store after pasteurization” could mean an isothermal hold at some moderately high T to extend shelf life (like 55°C). - Как param, PAR\_StoreEnabled=0 default, 1 optional. Possibly if 1, they treat SP\_CoolTemp as “storage temperature” to hold product until next step. If so: - PasteurMgr: после HoldTime, не открывать сразу охлаждение, а перейти в “Store” stage: - EN\_Heat stays TRUE? If store at pasteurization temp? or at a lower store temp? - There’s no separate SP for store given (they might reuse SP\_CoolTemp as store temp if store requires a lower hold). - Could be: - If StoreEnabled=1 and SP\_CoolTemp > 0: treat SP\_CoolTemp as new target after hold, gradually cool to it and then maintain it (like extended hold at lower T). - Actually, code for CIP store not given, likely not fully implemented. - Alternatively, store after pasteurization might mean they keep pasteurization conditions until product is needed. But not typical – usually want to cool. - Considering default StoreEnabled=0, we suspect it’s not used. - *Проектное решение:* - Если StoreEnabled=1: после Cool stage, PasteurMgr не завершает работу, а держит режим: - EN\_Heat: if need to maintain (maybe if product starts to drop below SP\_CoolTemp by a margin, could re-enable heat). - Could incorporate small hysteresis around SP\_CoolTemp: if product < SP\_CoolTemp - 2°, turn EN\_Heat TRUE to reheat up to CoolTemp; if > +2°, turn off. In effect maintain around SP\_CoolTemp. This is basically thermostatic storage. - This effectively toggles between heat and cool phases as needed to keep stable. - Possibly mixers and pumps remain on entire time to uniform. - The cycle would then only stop on operator command (like Start Dispense triggers end of store or abort CIP triggers flush). - In absence of clarity, we mark [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ] for store logic. For TЗ, main focus is to highlight that store mode must ensure safe product hold, which may involve periodic heating. - Если StoreEnabled=0 (обычно): PasteurMgr по достижении CoolTemp завершает цикл: - Выдает сигнал окончания (STEP\_Pasteur = code, e.g., 4 “Done”). - Сбрасывает RunReq (Mode\_Pasteur\_RunReq->FALSE – Main возможно). - Отключает циркуляцию, мешалку (CircPump\_ON=FALSE, Mixer\_ON=FALSE, CoolingValve=FALSE). - Устанавливает флаг Mode\_PasteurActive=FALSE (Main тоже). - ActuatorManager, увидев снятие PasteurActive, выключит соответствующие DO. - Теперь система готова либо к розливу, либо просто ждет.

* **Преждевременный останов (Abort):** Если оператор нажал Stop Pasteur (MB\_CmdMask1.b1) или случилась критическая авария (E-Stop, Sensor fail), PasteurMgr должен безопасно остановиться:
* Если в фазе Heat/Hold: лучше сначала выключить нагрев (EN\_Heat=FALSE), включить охлаждение (если возможно, чтобы быстро сбить температуру – но может и не нужно, можно просто остановиться, продукт останется горячим).
* Остановить таймеры.
* Установить флаг ALM\_PasteurAbort (бит13), чтобы оператор увидел, что пастеризация не завершена корректно.
* Перейти к Idle: выключить мешалку, оставить может циркуляцию какое-то время? (Better, открыть Drain if needed? No, drain is CIP context).
* Так как продукт непастеризован (не выдержан), возможно, его надо будет слить или перепастеризовать – но это вне автоматики.
* Код: Main или PasteurMgr должны прервать RunReq.
* Actuator all off by ActuatorMgr if E-Stop, or by logic if no longer active.
* Already present ALM\_PasteurAbort var suggests, set it and mark cycle ended aborted.
* CIP similarly has ALM\_CIPAbort (bit14) for CIP abort scenario.
* **Выходные сигналы PasteurManager:** (уже видели VarOutput)

| Signal | Dir | Type | Unit | Consumers | Description | Default | Retain |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **CircPump\_ON** | Out | BOOL | – | ActuatorMgr.In【30†】 | Включить циркуляционный насос рубашки. TRUE на всех этапах Pasteur, где нужен теплоноситель (нагрев, охлаждение, возможно хранение). | FALSE | No |
| **Mixer\_ON** | Out | BOOL | – | ActuatorMgr.In【107†】 | Включить мешалку продукта. Обычно TRUE на всём протяжении пастеризации (для равномерности). | FALSE | No |
| **CoolingValve\_ON** | Out | BOOL | – | ActuatorMgr.In【42†】 | Открыть клапан охлаждения рубашки. TRUE на этапе охлаждения (Cool), возможно на хранении (если нужно поддерживать T?). Управляется с гистерезисом от рубашки【79†】. | FALSE | No |
| **HeatingPowerReq** | Out | BOOL | – | FB\_Main/TempMgr/ActuatorMgr | **Примечание:** В коде указан как BOOL “EN\_Heat”【59†】, фактически это “разрешение нагрева”. ActuatorMgr не использует напрямую, но Main присваивает TempMgr.EN\_Heat = this. | FALSE | No |
| **STEP\_Pasteur** | Out | UINT | code | HMI (MB 545) | Код текущего этапа пастеризации: 0=Idle, 1=Heat, 2=Hold, 3=Cool, 4=Done, 5=Abort. Обновляется PasteurMgr при переходах. | 0 | No |
| **PasteurActive** (возможно) | Out | BOOL | – | FB\_Main? Actuator? | В некоторых проектах FB может сам ставить глобальный флаг активность. Но у нас глобал MODE\_PasteurActive ведется Main. PasteurMgr может косвенно (через RunReq logic) – опустим. | – | – |

PasteurManager хранит внутренние таймеры: - t\_hold – остаток/счетчик выдержки (UDINT). - t\_heat – таймер контроля разгона (может, на основе Seconds\_UD). - stage – текущее состояние (enum или int). - SkipHeatCmdLatched – флаг, что был получен SkipHeat (чтобы не среагировать дважды). - Flags for abort etc.

**Таблица состояний PasteurManager:**

| Stage | Условия входа | Действия (Outputs) | Условия перехода |
| --- | --- | --- | --- |
| Idle (0) | RunReq = FALSE | Все выходы FALSE (нагрев выкл, охлаждение выкл, мешалка выкл). | RunReq TRUE получен -> Stage=Heat. |
| Heat (1) | RunReq TRUE (начало цикла пастер.) | HeatingPowerReq=TRUE (EN\_Heat), CircPump\_ON=TRUE, Mixer\_ON=TRUE, CoolingValve\_ON=FALSE. Запуск нагрева. Запуск таймера разгона (если SP\_HeatTime>0). | Если PV\_TempProduct >= SP\_HeatTemp или SkipHeatCmd -> Stage=Hold. <br>Если истек HeatTime -> приостановка (см. ниже). <br>Если StopCmd -> Stage=Abort. |
| Hold (2) | Условие: достижение Temp или skip | (EN\_Heat остаётся TRUE), продолжить CircPump\_ON=TRUE, Mixer\_ON=TRUE, CoolingValve\_ON=FALSE. Запустить таймер выдержки = SP\_HoldTime. | По истечении HoldTime -> Stage=Cool. <br>Если StopCmd/E-Stop -> Abort. |
| Cool (3) | По окончании выдержки | EN\_Heat=FALSE (нагрев выкл), CircPump\_ON=TRUE (продолжает циркуляцию при охлаждении), Mixer\_ON=TRUE (помешивание), CoolingValve\_ON=TRUE (открыть холод). Начать мониторить достижение SP\_CoolTemp. | Если PV\_Product <= SP\_CoolTemp (или SP\_CoolTemp=0) -> либо Stage=Done либо Stage=Store (если StoreEnabled). <br>StopCmd -> Abort (хотя уже почти конец, можно просто Done). |
| Store (4) [опц] | StoreEnabled=TRUE | **Вариант 1:** EN\_Heat=FALSE, CircPump\_ON может FALSE (если рубашка уже охлаждена), Mixer\_ON= возможно TRUE, CoolingValve\_ON=FALSE (если уже охладили). Просто удержание продукта без охлаждения (продукт уже на нужной temp). <br>**Вариант 2:** EN\_Heat=TRUE периодически, чтобы поддерживать Temp (если нужно держать теплым). Может реализовываться управлением TempMgr: например, при падении ниже SP\_CoolTemp-Δ включать EN\_Heat, при превышении на Δ выключать. | Ожидание команды оператора (обычно запуск режима розлива) или лимит времени хранения? В текущем проекте, скорее выход из Store по внешней команде: либо Stop (Abort) либо переход в Dispense (который сам требует PasteurActive=FALSE). Скорее, оператор нажмет Start Dispense, что деактивирует Pasteur RunReq. Тогда Stage->Done. |
| Done (5) | По завершении охлажд/хранения | Все выходы = FALSE (на случай, если что-то еще активно). Фиксирует завершение цикла. Может установить бит “Pasteur complete” где-то (не требуется). | Здесь RunReq будет снят (Main). Переход в Idle автоматически либо просто конец. |
| Abort (6) | По команде стоп или аварии | EN\_Heat=FALSE, CoolingValve\_ON=TRUE? (спорно), CircPump\_ON возможно TRUE еще неск.сек (чтобы отвести тепло), потом FALSE, Mixer\_ON=FALSE (чтобы прекратить механическое воздействие? Либо оставить мешать чтоб не пригорело? Можно оставить пока рубашка горячая). Устанавливает ALM\_PasteurAbort=TRUE. | После выполнения аварийных действий – RunReq сбрасывается, Mode\_PasteurActive=FALSE (Main), ActuatorMgr выключит все. Переход Idle. |

*(Stage codes 4/5/6 hypothetical, actual STEP might code differently. Possibly they used 0-4 or 0-3 code only.)*

**Особенности:** - **CircPump\_ON:** можно выключить на Storage или Done, но, вероятно, PasteurMgr держит его TRUE до конца Cool, а затем OFF. - **Mixer\_ON:** возможно, можно выключить после Hold, если охлаждение идет внешней рубашкой – но лучше мешать до конца охлаждения. Probably off at Done. - **CoolingValve\_ON:** toggled in Cool with hysteresis as per code【79†】. So PasteurMgr likely runs code in Cool: if Jacket <= SP\_CoolTemp - Diff then close valve (to not overcool). That is inside CIPActive check for CIP, but here used in pasteur code snippet, which indeed shows that logic (though comment says "защита от переохлаждения рубашки", likely meaning when product nearly reached target, to not continue cooling). - **HeatTime limit:** If triggered, not defined what to do in code. Possibly remain in Heat stage waiting for skip or abort. - They may expect operator either increases SP\_HeatTime (commit delay?), or decides to skip heat stage. They had CIP\_LiveWriteCommitDelay for CIP editing. Not for pasteur. - Possibly, if HeatTimeout triggers, they raise STAT\_HeatTimeout (Ram status) and do nothing else. HMI sees it (maybe blinking?) and operator can press SkipHeat to proceed anyway. So: - When STAT\_HeatTimeout=TRUE, Main might also set MB\_SystemStatus.b3 "Reset required"? Or some indicator. - If operator chooses, they press SkipHeat (b2) to skip to hold anyway with whatever temperature reached. - Or Stop to abort. - After skip, ALM\_PasteurAbort? or just continue? It's not a normal situation, but maybe treat skip as manual override, not an alarm.

**Интерфейс с Main:** - Main sets Mode\_Pasteur\_RunReq for PasteurManager.RunReq. - PasteurManager outputs influence Actuator (valves, pump). - PasteurManager should ideally inform when done (Main sees Mode\_Pasteur\_RunReq=FALSE if done). - Possibly Main monitors PasteurActive and resets reqPasteur or triggers next modes.

**Итоги по PasteurManager:** - Инварианты: - Во время PasteurActive, EN\_Heat=TRUE всегда, кроме этапа Cool/Store. - DO1 (Cooling) = !EN\_Heat (по фазам). - DO2 (CircPump) = always ON while PasteurActive. - DO3 (Mixer) = always ON while PasteurActive. - Safe-state: При Abort, все приводы либо выключены, либо (CircPump может оставаться на 30с) – [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ].

### 2.5 FB\_CIPManager\_v2 – цикл CIP мойки

**Назначение:** Реализует автоматический цикл CIP (Clean-in-Place) очистки емкости/коммуникаций пастеризатора. Поддерживает несколько предустановленных режимов (Full – полный цикл, Alk – щелочной, Acid – кислотный, Rinse – ополаскивание) путем комбинации до 7 последовательных циклов. Каждый цикл (этап CIP) описывается набором параметров: длительности подачи различных сред (горячая/холодная вода, щелочь, кислота), длительность циркуляции насоса, длительность слива. CIPManager активирует соответствующие клапаны и насос CIP в нужное время, следит за таймингами, контролирует последовательность сред (в соответствии с выбранным режимом и политикой конфликтов щел./кисл.). Обрабатывает прерывание (Stop CIP) с безопасным сливом остатка.

**Описание режимов CIP:** - **Full** – полный цикл из 7 этапов: обычно включает все стадии: 1. Предополаскивание (Cold + Hot water flush), 2. Щелочная мойка, 3. Промежуточное ополаскивание, 4. Кислотная мойка, 5. Промежуточное ополаскивание, 6. Финальное ополаскивание (чистой водой), 7. Дренирование. (пример, может отличаться). Исходя из маски Full=127 (b0-6=1) – все 7 циклов выполняются. - **Alk** – щелочной: включает этапы 1,2,3,6,7 (маска 103 dec = b0,b1,b2,b5,b6=1) – т.е. пропускается кислотная мойка (4,5). - **Acid** – кислотный: включает 1,2,5,6,7 (mask 115 = b0,b1,b4,b5,b6=1) – пропускается щелочная (циклы 2,3?). Подозрение: mask bits 4,5 correspond to cycles 5,6? Actually if binary 115, bits: 0,1,4,5,6 – that's cycles 1,2,5,6,7; indeed skipping cycles 3,4 (post-alk flush and acid wash). - **Rinse** – ополаскивание: включает только последние 2 этапа (6,7) – mask 96 dec = b5,b6=1.

Каждый **цикл** CIP (номер 1..7) имеет в настройках: - CIP\_C#\_T\_HOT\_s – время подачи горячей воды (сек). - CIP\_C#\_T\_COLD\_s – время подачи холодной воды. - CIP\_C#\_T\_ALK\_s – время подачи щелочи. - CIP\_C#\_T\_ACID\_s – время подачи кислоты. - CIP\_C#\_T\_PUMP\_s – время циркуляции (обычно когда не подается новая среда, но насос гоняет по кругу). - CIP\_C#\_T\_DRAIN\_s – время слива.

Не все этапы используют все типы сред: - Например, этап1 (предопл.) может задать T\_COLD и T\_HOT >0, а ALK/ACID =0. CIPManager тогда в цикл1 последовательно выполнит: открыть Cold клапан на T\_COLD, затем (возможно) Hot клапан на T\_HOT (или одновременно? Скорее последовательно, т.к. один насос). - Этап2 (щелоч): T\_ALK>0, T\_HOT возможен тоже (после щелочи обычно горячая вода?), или T\_PUMP =0? - В общем, конфигурация гибкая. Но вероятно: - Cycle1: flush – Cold + Hot, - Cycle2: Alk – Alk + circulation + drain, - Cycle3: rinse after alk – Cold or Hot flush, - Cycle4: Acid – Acid + circ + drain, - Cycle5: rinse after acid – flush, - Cycle6: final rinse – flush thoroughly, - Cycle7: drain final.

В Retain-переменных (CSV) заданы все CIP\_C#*T*\* (retain).

**Редактирование параметров CIP:** - Так как много параметров, UI сделано через регистры MB\_CIP\_Edit\_ *(530-533), CIP\_Edit\_Select, CIP\_LiveWriteCommitDelay. - Оператор выбирает номер цикла (Select = 1..7), панель читает retain значения CIP\_C# for that cycle и заполняет MB\_CIP\_Edit\_* registers. - После правки, по истечении CIP\_LiveWriteCommitDelay (по умолч 2с), CIPManager копирует MB\_CIP\_Edit\_ *обратно в CIP\_C#\_* retain (сохраняя изменения)【81† комментарий】. - Это позволяет редактировать время щелочи, воды и т.д. вне выполнения или даже на паузе. - CIP\_LiveWriteCommitDelay, CIP\_ChemConflictPolicy – используются CIPManager: - Если Alk и Acid оба выбраны (Full), conflictPolicy=1 (пр. ALK) – вероятно, определяет, что делать если Alk+Acid mode launched? But in modes design, Alk and Acid are separate modes, conflict not possible except user might try to start two CIP modes concurrently (ActiveMode prevents that). - Perhaps if user accidentally sets both CIP Alk and CIP Acid bits (b7 & b8) at once, conflict policy decides which to run. But ActiveMode logic in Main checks CIPFull, then Alk, then Acid, then Rinse sequentially (only one). - Possibly conflict policy relevant if operator manually toggles bits to include both chemicals in one custom run? But they already have Full. - We can mention: if conflict=0, CIP refuse to run if both Alk & Acid cycles included; if 1 – allow Alk, skip Acid; if 2 – allow Acid, skip Alk. In Full mode, both included by design (so conflict?), but full presumably rinse in between so conflict avoided. - Actually maybe conflict refers to if cleaning solutions inadvertently could mix (like if user accidentally config cycle with both Alk and Acid times >0 in same stage). - CIPManager might ensure not to open Alk and Acid simultaneously no matter what times given (likely sequential). - We'll state as parameter for completeness.

**Интерфейс CIPManager:**

| Signal (FB\_CIPManager\_v2) | Dir | Type | Unit | Source/Consumers | Description | Default | Retain |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **CIP\_Full\_RunReq** | In | BOOL | – | FB\_Main.Mode\_CIP\_Full\_RunReq【48†】 | Запрос запуска полного CIP. Если TRUE, CIPManager начинает программу Full. | FALSE | No |
| **CIP\_Alk\_RunReq** | In | BOOL | – | FB\_Main.Mode\_CIP\_Alk\_RunReq【48†】 | Запуск щелочного CIP. | FALSE | No |
| **CIP\_Acid\_RunReq** | In | BOOL | – | FB\_Main.Mode\_CIP\_Acid\_RunReq【48†】 | Запуск кислотного CIP. | FALSE | No |
| **CIP\_Rinse\_RunReq** | In | BOOL | – | FB\_Main.Mode\_CIP\_Rinse\_RunReq【48†】 | Запуск ополаскивания CIP. | FALSE | No |
| *(Примечание: только один из выше может быть TRUE одновременно)* |  |  |  |  |  |  |  |
| **CIP\_ModeMask\_Full\_IN** | In | UINT | bits | Retain (127) | Маска этапов для режима Full (биты 0-6). Обычно =127 (все этапы). CIPManager может использовать для вывода CIPStagesMask. | 127 | Yes |
| **CIP\_ModeMask\_Alk\_IN** | In | UINT | bits | Retain (103) | Маска этапов для режима Alk. По умолчанию 103. | 103 | Yes |
| **CIP\_ModeMask\_Acid\_IN** | In | UINT | bits | Retain (115) | Маска этапов для Acid (115). | 115 | Yes |
| **CIP\_ModeMask\_Rinse\_IN** | In | UINT | bits | Retain (96) | Маска этапов для Rinse (96). | 96 | Yes |
| **CIP\_C1\_T\_HOT\_IN** ... **C1\_T\_DRAIN\_IN** | In | UDINT | s | Retain | Набор параметров цикла 1 (Times). (Analogично для C2..C7). CIPMgr читает их для выполнения. | – | Yes |
| **MB\_CIP\_Edit\_T\_COLD** | In | UDINT | s | Modbus 530 (HMI)【8†】 | Текущее значение, редактируемое оператором, времени холодной воды для выбранного цикла. CIPMgr копирует эти значения в *\_IN/*\_OUT при commit. | – | No |
| **… MB\_CIP\_Edit\_T\_ALK, T\_ACID, T\_PUMP, T\_DRAIN** | In | UDINT | s | Modbus 531-533 | Аналогично – редактируемые значения. | – | No |
| **MB\_CIP\_Edit\_Select** | In | UINT | – | Modbus (539 или 569/570?) | Номер выбранного цикла для редактирования (1-7). CIPManager использует, чтобы знать, в какие CIP\_C#\_ *сохранять MB\_CIP\_Edit\_* при commit. | – | No |
| **MB\_CIP\_Commit** | In | BOOL | – | Possibly reg.570 bit? | Сигнал commit (возможно, если нужна ручная фиксация). Здесь, скорее, commit авт. через таймер CIP\_LiveWriteCommitDelay. | – | No |
| **CIP\_LiveWriteCommitDelay\_s** | In | UDINT | s | Retain (2) | Время задержки перед сохранением изменений параметров CIP. CIPMgr: если MB\_CIP\_Edit\_\* изменились и не меняются больше этого времени – выполнить commit (копирование в retain). | 2 | Yes |
| **CIP\_ChemConflictPolicy** | In | UDINT | code | Retain (1) | Политика конфликта реагентов: 0=запрет, 1=приоритет Alk, 2=приоритет Acid, если одновременно заданы. Используется, если Alk & Acid циклы пересекаются (в режиме Full). | 1 | Yes |
| **DI6\_ProductFlow** | In | BOOL | – | FB\_Main (DI6) | Датчик потока/наличия жидкости (НО). 1=поток есть. Используется CIPMgr для контроля: если 0 слишком долго во время подачи – ALM\_NoWater. | – | No |
| **Man9\_ValveCIP\_Cold, Man10\_ValveCIP\_Hot, Man11\_ValveAlk, Man12\_ValveAcid, Man13\_PumpCIP, Man14\_ValveDrain** | In | BOOL | – | FB\_Main (Manual bits) | Ручные команды CIP. **В CIPManager они не используются напрямую** (идут сразу в Actuator). Но CIPMgr может отслеживать, если ManualActive=TRUE, и останавливать CIP. | FALSE | No |
| **Mode\_CIP\_ACTIVE** | In | BOOL | – | (from Main) | Флаг, что CIP режим активен (для инф, ActuatorMgr). CIPMgr, возможно, не нужен как вход, он сам знает, когда RunReq. | FALSE | No |

**Выходы CIPManager:**

| Signal | Dir | Type | Unit | Consumers | Description | Default | Retain |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **CIP\_Valve\_Cold\_ON** | Out | BOOL | – | ActuatorMgr.In【30†】 | Команда открыть клапан холодной воды CIP (DO9). Включается на промежутки времени T\_COLD этапов, где он >0. | FALSE | No |
| **CIP\_Valve\_Hot\_ON** | Out | BOOL | – | ActuatorMgr.In | Открыть клапан горячей воды CIP (DO10). Включается на промежутки T\_HOT. | FALSE | No |
| **CIP\_Valve\_Alk\_ON** | Out | BOOL | – | ActuatorMgr.In | Открыть клапан подачи щелочи (DO11) на T\_ALK. | FALSE | No |
| **CIP\_Valve\_Acid\_ON** | Out | BOOL | – | ActuatorMgr.In | Открыть клапан подачи кислоты (DO12) на T\_ACID. | FALSE | No |
| **CIP\_Pump\_ON** | Out | BOOL | – | ActuatorMgr.In | Включить насос CIP (DO13). В большинстве этапов работает постоянно: включается в начале этапа и выключается только на дренировании (или стопе). Может отключаться на этапах flush? Но скорее постоянно. | FALSE | No |
| **CIP\_DrainValve\_ON** | Out | BOOL | – | ActuatorMgr.In | Открыть клапан слива (DO14). Включается на фазах DRAIN (конец каждого цикла на T\_DRAIN). | FALSE | No |
| **CIP\_Mixer\_ON** | Out | BOOL | – | ActuatorMgr.In【30†】 | Включить мешалку во время CIP. Если CIP\_ModeMask\_xxx предполагает мешалку на определенных этапах (в CSV есть CIP\_C1\_MixerStage1 etc. – скорее boolean использовать мешалку на этапе1). Retain CIP\_C#\_MixerStageX (CSV shows CIP\_C1\_MixerStage1 etc. boolean). Если они установлены, CIPMgr выставляет CIP\_Mixer\_ON=TRUE на соответствующем подэтапе. Это нужно, чтобы промывать стенки. | FALSE | Yes (set per cycle) |
| **MB\_CIPStagesMask** | Out | UINT | bits | HMI (reg.515) | Маска этапов, включенных в текущий выполняемый режим CIP. Когда CIP запущен, CIPMgr записывает сюда выбранную маску (Full/Alk/…). Используется HMI для отображения последовательности. | 0 | No |
| **STEP\_CIP** | Out | UINT | code | HMI (reg.546) | Код текущего шага CIP: 0=Idle, 1–7 = номер цикла (этап), возможно 8=Done, 9=Abort. Обновляется CIPMgr по мере перехода между циклами. | 0 | No |
| **ALM\_NoWater** | Out | BOOL | – | Global Alarm (Ram) | Авария: нет потока/жидкости. Если DI6=0 в течение ожидаемого наличия потока (например, когда открыт любой клапан и насос работает), CIPMgr через несколько секунд ставит ALM\_NoWater=TRUE и переходит в Abort. | FALSE | No |
| **CIPAbortFlag** (внутр.? or ALM) | Out | BOOL | – | Possibly ALM\_CIPAbort (bit14) | Если CIP прерван (StopCmd/E-Stop), CIPMgr может установить ALM\_CIPAbort = TRUE (bit14). А также MB\_Alarms будет содержать. | FALSE | No |

**Работа CIPManager:** - **При старте:** При входе одного из RunReq (Full/Alk/Acid/Rinse), CIPMgr: - Выбирает активную маску: напр., если CIP\_Full\_RunReq=TRUE, берет CIP\_ModeMask\_Full\_IN (retain) как Mask. - Устанавливает глобальный MB\_CIPStagesMask = Mask (HMI видит какие этапы будут). - Reset internal cycle counter to first bit set in mask (likely 1). - Включает насос CIP: CIP\_Pump\_ON = TRUE (всегда, с начала до конца CIP; либо он может выключаться на дренажных паузах – но скорее пусть работает непрерывно, подавая жидкость/циркулируя, и на сливе – может отключать, чтобы насос в сухую не молотил). - В принципе, лучше: - На фазах подачи/циркуляции – насос ON, - На фазах слива – насос OFF (чтобы дать стечь; либо можно оставить, чтоб помогал выгонять жидкость? Обычно на слив – отключают насос, открывают дренаж). - CIPMgr будет работать как state machine: - For each Cycle i from 1 to 7: - If Mask bit i = 0: пропустить (но, если skip, ensure to do something? likely just skip to next). - If Mask bit i = 1: выполнить: - Set STEP\_CIP = i (for display). - Possibly, MB\_CIPStagesMask bits or separate indicator for current stage (maybe not needed). - Execute sub-stages within cycle i: 1. **Flush stage:** If T\_COLD > 0: - Open ValveCold, set CIP\_Valve\_Cold\_ON=TRUE, run for T\_COLD sec. - Close ValveCold. - If T\_HOT > 0: - Open ValveHot CIP\_Valve\_Hot\_ON=TRUE for T\_HOT sec. - Close ValveHot. - If T\_ALK > 0: - Open ValveAlk CIP\_Valve\_Alk\_ON=TRUE for T\_ALK sec. - Close ValveAlk. - If T\_ACID > 0: - Open ValveAcid CIP\_Valve\_Acid\_ON=TRUE for T\_ACID sec. - Close ValveAcid. - (Note: The sequence of these mediums in one cycle might be predetermined or only one of them is non-zero typically. But possibly can configure multiple in one cycle as an advanced user – then CIPMgr will apply sequentially). 2. **Circulation stage:** After adding those mediums, if T\_PUMP > 0: - Ensure all chemical valves closed (already did), - Keep CIP\_Pump\_ON running (already on by default; maybe ensure it stays), - Possibly, if CIP\_MixerStageX for this stage = TRUE, set CIP\_Mixer\_ON=TRUE (to stir the tank during circulation) for T\_PUMP sec【81† comment】. - After T\_PUMP sec, CIP\_Mixer\_ON=FALSE (stop stirring). 3. **Drain stage:** If T\_DRAIN > 0: - Stop adding mediums (already done), - Turn off CIP\_Pump\_ON? Possibly yes, to let drain. Alternatively, keep pump on to pump out (but pumping out through drain doesn't make sense if drain is bottom valve, the pump typically draws from bottom and outputs to cleaning loop, not to drain). - Likely: close CIP\_Pump, open CIP\_DrainValve\_ON=TRUE for T\_DRAIN sec (let gravity drain out), - After T\_DRAIN, close DrainValve. - If CIP\_Pump was off, turn it back on after drain if there are more cycles (to continue). - Or another approach: some CIP systems route pump output to drain by switching valves; but here likely drain is separate path not using pump to avoid contamination of pump with chemicals? Hard to say. We'll assume drain is gravity drain from lowest point. - End of cycle i. - Then proceed to next cycle bit. - After last cycle: - Keep CIP\_Pump running for any leftover flush? Actually, presumably in final drain (cycle7) we turned pump off anyway to let final drain. So after CIP done, pump is likely off, all valves closed. - CIPCompleted: STEP\_CIP -> maybe 8 or 0, MB\_CIPStagesMask maybe cleared or left. - Mode\_CIPActive to FALSE (Main handles that by RunReq false). - Possibly flush signals ALM if needed. - CIP\_ChemConflictPolicy: - If user inadvertently had Alk & Acid mediums in same run: - If policy=0, CIPMgr could abort or skip both? But probably skip acid if both present since Full covers both with rinses. - Actually conflict could mean in Mode Full if something goes wrong with one chemical supply, use other? not exactly. - Possibly it's meant if user tries to launch Alk CIP right after Acid CIP (immediately or concurrently?), but ActiveMode prevents concurrent. - Honestly, conflict policy unclear; perhaps skip.

* **Stop CIP (Abort):**
* If MB\_CmdMask2.b10 (CIP Stop) is triggered during CIP:
  + CIPMgr should abort current cycle:
  + Immediately stop adding chemicals (close valves),
  + If it's safe, perform a quick drain:
    - Open DrainValve for CIP\_DrainOnStopSec (retain param, default 30s).
    - Keep CIP\_Pump on for a short while if it helps push fluid out, or off to let gravity (prefer off to not fight).
  + After that, close everything, stop pump.
  + Set ALM\_CIPAbort=TRUE (bit14).
  + Exit CIP (RunReq false).
  + ActuatorMgr will handle outputs off.
* If E-Stop triggered: ActuatorMgr anyway kills all DO (pump off etc). CIPMgr might not even get chance to gracefully drain (unless battery backup or after restart). But ideally:
  + If E-Stop, output should be safe state: pump off, valves closed except maybe drain if stuck open? But since E-Stop likely kills power, not relevant. We'll skip detail.
* **DI6\_NoFlow monitoring:**
* CIP process expects flow when medium valves open and pump on. If DI6 remains 0 for e.g. >5s after starting flush or circulate, it indicates no fluid (maybe tank empty or pump lost prime). CIPMgr then triggers ALM\_NoWater (lack of medium flow) and aborts CIP to avoid dry run or no cleaning.
* Could implement a check each time it opens a medium valve:
  + Start timer, if DI6 doesn't go 1 within e.g. 2s, ALM\_NoWater.
  + Or continuously monitor DI6, if it becomes 0 for >X sec while CIPActive (particularly when pump on and not draining), raise ALM.
* As result, CIP stops pump, abort, maybe open drain to avoid chemical sitting.
* ALM\_NoWater is separate from ALM\_Mask bits (since ALM\_Mask covers 0-15, maybe NoWater is beyond). Actually ALM\_NoWater listed outside mask possibly, or might not included or it might correspond to bit, unclear.
  + It's listed after ALM\_Mask and before AO\_HeaterPower in CSV, presumably meaning it’s not in the 16-bit mask. Possibly a separate alarm not shown on HMI mask or to be ORed into ALM\_Mask if needed.
  + We'll treat it as separate global alarm flag.

**Инварианты CIP:** - At most one CIP mode runs at once. - CIP valves (Cold/Hot/Alk/Acid) never open simultaneously; CIPMgr sequences them. Possibly Cold and Hot might both be used in one cycle but sequentially, not overlapping (to not mix temp). - CIP Pump ideally on whenever a fluid is circulating (except drains). - CIP Drain only open when pump off (to allow gravity). - CIPMixerOn toggled only if configured, ensures to run only when liquid in tank (not on drain step presumably).

**Выход CIP signals and Actuator logic**: - ActuatorMgr uses CIP\_x\_ON only if Mode\_CIP\_ACTIVE (ensures not affecting other modes)【42†】. - CIP\_Mixer\_ON goes to DO3 forward during CIPActive【107†】.

### 2.6 FB\_DosingManager – режимы розлива

**Назначение:** Управляет процессом розлива продукта из емкости пастеризатора либо непрерывно (по трубопроводу) либо порционно (дозируя заданный объем). Обеспечивает взаимодействие с оператором: учет сигнала “пистолет на месте”, кнопки порционного розлива, сигнализирует окончание дозы. Реализует простой программный счетчик объема (через SoftFlowmeter). Гарантирует, что при отсутствии разрешающих условий (пистолет снят, кнопка не нажата) порция не начнется.

**Режимы:** - **Continuous dispense:** просто включает насос розлива и держит его включенным, пока режим активен (как насос опорожнения). Оператор вручную управляет старт/стоп (кнопками Start/Stop DispCont). Этот режим не требует дозирования; DosingManager в нем не следит за объемом. - **Dose dispense:** при активации (Start DispDose) – насос включается порционно: т.е. DosingMgr наливает одну порцию заданного объема, затем автоматически останавливает насос и ждет команды на следующую порцию. Если параметр PAR\_UsePortionButton=FALSE, то порции могут идти подряд автоматически (DosingMgr сам сразу начинает следующую порцию после небольшой задержки). Если PAR\_UsePortionButton=TRUE, то после каждой порции DosingMgr ждет нажатия внешней кнопки (DI4) или команды NextDose (HMI) для старта следующей. - Обе режима – **взаимоисключающие** (т.е. Main подает EN\_Continuous или EN\_Portion). Если оба RunReq=FALSE – DosingMgr idle (насос выкл.).

**Условия запуска:** - Режимы розлива включаются **только** когда нет пастеризации/CIP. Main позаботится. - При начале Dispense, желательно чтобы продукт был пастеризован. Если DispDose – возможно, продукт сначала накапливается, потом дозируется. - DosingManager сам не контролирует температурные или другие условия, предполагается, что пастеризация завершена (PasteurActive=FALSE) к этому моменту, или DispenseActive=TRUE. - Однако, DosingMgr может быть активирован даже если PasteurActive (произойдет conflict, Main should avoid). - **Interlocks:** - Если PAR\_UseGunSensor=TRUE, DosingMgr не позволит включить насос, пока GunSensor=0 (пистолет на месте). В CSV DI5 "GunSensor (NO):1=снят", значит GunSensor bool вероятно = ~NOT(DI5) (we can define differently). - Если PAR\_UsePortionButton=TRUE (для порционного): DosingMgr не запускает порцию автоматически, а ждет внешней кнопки. - **Stop conditions:** - Если режим DispenseRunReq=FALSE (оператор остановил режим) – DosingMgr немедленно останавливает насос (если лил порцию – прерывает). - E-Stop – Actuator off, so pump stops anyway. - Gun sensor removal mid-pour (if PAR\_UseGunSensor=TRUE) – DosingMgr может остановить насос (т.к. пистолет сняли, нужно прекратить). В требованиях: “разрешать розлив только при пистолете на месте” implies if sensor goes false (no gun), likely stop pumping. DosingMgr нужно мониторить GunSensor continuously, не только перед стартом: - Implementation: if GunSensor goes FALSE (meaning gun removed mid-run?), then stop pump and maybe wait until placed back to resume? - Possibly treat removal mid-dose as abort of dose: so pump off, portion incomplete, user might restart portion or do something. - We'll specify: if gun sensor lost, pump off, portion aborted (no portion done signal). - If gun returns and portion button pressed again, probably new portion from scratch. - Alternatively, could pause portion (not implement).

**Связь с SoftFlowmeter:** - SoftFlowmeter (see 2.8) integrates flow to estimate volume pumped. - DosingMgr, при начале каждой порции: - Сбрасывает счетчик (SoftFlowmeter.Reset=TRUE на один цикл). - Выставляет SoftFlow.Enable=TRUE, PumpOn=TRUE (SoftFlow counts). - Постоянно опрашивает SoftFlow.Volume (VolumeTotal). - Когда Volume >= SP\_DoseVolume, DosingMgr считает порцию законченной: - отключает насос (FillPump\_ON = FALSE). - Устанавливает флаг PortionDone = TRUE (импульс). - PortionDone побудет TRUE один цикл (или пока Main/HMI увидит). - Если PAR\_UsePortionButton=FALSE: через, скажем, 0.5 секунды (или сразу на след.цикле) DosingMgr автоматически начинает новую порцию (если DispenseRunReq ещё TRUE). То есть reset Volume, PumpOn=TRUE снова. - Если PAR\_UsePortionButton=TRUE: DosingMgr ждет внешней команды – не включает насос снова, пока не нажата DI4 или NextDose (Mask1.b8). Как увидеть: DosingMgr может мониторить DI4 rising edge (PortionButton). - DI4 is NO (1 when pressed), likely need edge detect to avoid continuous start if held. - Also NextDose from HMI triggers Mode\_NextDose or something (we have CMD\_NextDose in Commands bits, bit8). - Implementation: DosingMgr sets an internal flag e.g. awaitTrigger = TRUE after finishing portion if UsePortionButton=TRUE. - It then monitors: if DI4 goes 1 or CMD\_NextDose bit toggled (Main could set a global CMD\_NextDose flag; we have global CMD\_NextDose in commands list). - On trigger, it resets Volume and starts pump for next portion. - Continue until Mode\_DispDose\_RunReq goes FALSE or all portions done (no concept of predetermined number of portions). - Continuous mode: - SoftFlowmeter could still count volume, but DosingMgr likely ignores volume (maybe resets at start but doesn’t care). - It possibly resets volume at start for courtesy, but not needed. Or it can keep integrator disabled since not needed. Implementation: - If EN\_Continuous=TRUE: - FillPump\_ON = TRUE (immediately and continuously). - SoftFlow.Enable could be FALSE (not count), or can be enabled for monitoring total throughput if needed but not required by TЗ (maybe nice to display how many liters dispensed in total? Possibly not needed). - PortionDone flag not used in continuous.

**Интерфейс DosingManager:**

Inputs (some from earlier snippet【84†】): | Signal | Dir | Type | Source (from Main) | Description | Default | |------------------------|-----|------|--------------------------|--------------------------------------------------------------------|---------| | **EN\_Continuous** | In | BOOL | Mode\_DispCont\_RunReq | Запрос непрерывного розлива (уровневый). Когда TRUE, DosingMgr включает насос. | FALSE | | **EN\_Portion** | In | BOOL | Mode\_DispDose\_RunReq | Запрос порционного розлива. Когда TRUE, DosingMgr управляет насосом порциями. | FALSE | | **PAR\_UseGunSensor** | In | BOOL | FB\_Main (ParMask b3) | 1 = использовать датчик пистолета. Если TRUE, насос включается только при GunSensor=TRUE. | FALSE | | **GunSensor** | In | BOOL | FB\_Main (DI5, possibly inverted) | Состояние датчика пистолета. Нужно определить: 1 = пистолет на месте или снят? DI5 "НО:1=снят". Вероятно, здесь GunSensor = (NOT DI5) (т.е. TRUE = пистолет на месте). В коде snippet【98†】: canStart := canStart AND GunSensor after UseGunSensor check, implying GunSensor must be TRUE to allow start, значит TRUE=OK. Значит GunSensor = NOT(DI5). | FALSE | | **PAR\_UsePortionButton** | In | BOOL | FB\_Main (ParMask b2) | 1 = старт каждой порции только по внешней кнопке. | FALSE | | **PortionButton** | In | BOOL | FB\_Main (DI4) | Состояние кнопки порции (НО): 1 = нажата. DosingMgr отслеживает фронт 0->1. | FALSE | | **SP\_DoseVolume** | In | REAL | FB\_Main (global Var) | Объем одной порции, литры. Задается оператором (HMI) или рецептом. DosingMgr сравнивает с измеренным объемом. | 0.0 | | **PAR\_PumpFlowRate** | In | REAL | FB\_Main (retain Param) | Производительность насоса, л/с. Передается SoftFlowmeter. DosingMgr сам не использует, но может для debug. SoftFlow uses it. | 0.0 | | **CMD\_NextDose** | In | BOOL | FB\_Main (CMD, Mask1.b8) | Команда "Next Dose" с панели (вдобавок к кнопке). Main дублирует бит b8 (в CMD\_ Mask1) в Ram CMD\_NextDose. DosingMgr может воспринимать как виртуальное нажатие PortionButton. | FALSE |

Outputs: | Signal | Dir | Type | Consumers | Description | Default | |----------------------|-----|------|-------------------------------|--------------------------------------------------------------------|---------| | **FillPump\_ON** | Out | BOOL | ActuatorMgr.In【86†】 | Команда на включение насоса розлива (DO5). TRUE, когда нужно качать продукт. | FALSE | | **PortionDone** | Out | BOOL | HMI/FB\_Main (global)【95†】 | Импульс "Порция готова". Устанавливается в TRUE на один цикл после отмеривания порции. HMI может использовать для индикации/запуска внешних механизмов (например, переместить тару). | FALSE |

Внутри DosingManager: - FlowMeter (REAL) – переменная для общего накопленного объема, считывается из SoftFlowmeter (VolumeTotal). - dosingActive (BOOL) – внутренний флаг, что порция сейчас льется. - awaitTrigger (BOOL) – ожидание команды следующей порции (когда UsePortionButton=TRUE). - Possibly canStart (BOOL) – используется при проверке условий запуска: snippet【98†】 shows usage of canStart combining signals. - Timer or counters maybe.

**Логика DosingManager:**

Initialize: FillPump\_ON=FALSE, PortionDone=FALSE. SoftFlowmeter.Reset=FALSE, Enable=FALSE.

* **Continuous Mode (EN\_Continuous):**
* При EN\_Continuous становится TRUE (вход):
  + Если PAR\_UseGunSensor=TRUE и GunSensor=FALSE (пистолет снят) – **не включает насос**. DosingMgr может просто не включать FillPump\_ON, пока GunSensor false.
  + Если GunSensor TRUE или sensor disabled:
  + FillPump\_ON = TRUE (насос включен).
  + SoftFlow.Enable можно включить для мониторинга объема, но это не обязательно. Можно включить для интереса, VolumeTotal будет считать сколько налито непрерывно.
  + PortionDone в этом режиме не используется (не генерируется).
  + *Отличие от Portion mode:* continuous just one state: pumping or not.
* Пока EN\_Continuous=TRUE:
  + DosingMgr следит за GunSensor: если он становится FALSE (пистолет сняли) – *возможно*, остановить насос. Но continuous implies maybe they want to free flow? Actually, if gun removed means product pouring out freely, which is a hazard – so yes, likely must stop pump. So:
  + If PAR\_UseGunSensor=TRUE and GunSensor goes false, FillPump\_ON = FALSE.
  + Could wait until sensor true again and resume pump if still commanded.
  + But safer to require re-press to resume? Unclear. For now:
    - If during continuous dispensing gun removed (GunSensor false) -> treat as an emergency stop for dispensing: stop pump. Possibly remain in continuous mode but pump off until gun back and maybe automatically resume? That may surprise operator, maybe better not auto resume:
    - Possibly resume if gun placed back quickly. But likely not implemented in code, so:
    - We'll mention at least it stops. (In absence of requirements, we can decide to just stop and require re-toggle).
* При отмене режима (EN\_Continuous->FALSE):
  + FillPump\_ON = FALSE (if not already).
  + SoftFlow.Enable=FALSE (stop counting).
  + No portion done signal relevant.
* **Portion Mode (EN\_Portion):**
* Когда EN\_Portion становится TRUE:
  + Инициализация: DosingMgr готовится к первой порции.
  + awaitTrigger = FALSE initially.
  + If PAR\_UsePortionButton=TRUE:
  + Он **ждет команды**: не включает насос сразу.
  + В UI предполагается: оператор нажмет внешнюю кнопку, либо HMI NextDose, чтобы реально начать налив.
  + DosingMgr: canStart = (if PAR\_UsePortionButton then portionButton OR nextDose else TRUE).
  + Code snippet【98†】 suggests:
  + canStart := EN\_Portion; // presumably  
    IF PAR\_UsePortionButton THEN  
     canStart := canStart AND PortionButton;  
    END\_IF;  
    IF PAR\_UseGunSensor THEN  
     canStart := canStart AND GunSensor;  
    END\_IF;  
    IF canStart THEN  
     dosingActive := TRUE;  
    END\_IF;
  + This suggests:
    - They possibly treat portion start as immediate if portionButton pressed at mode start.
    - But likely they check these conditions continuously:
    - If EN\_Portion just turned on, portion not started yet (dosingActive=FALSE).
    - Each cycle, they check if conditions (PortionButton if needed, GunSensor if needed) are satisfied and if currently not in portion (dosingActive=FALSE and not awaiting next?), then set dosingActive=TRUE to start pump.
    - The code snippet indeed shows after combining canStart, they do IF canStart THEN dosingActive := TRUE.
    - So they might not explicitly wait for rising edge, just require button held at the moment mode turned on or if not, then portion won't start until user toggles something (less robust).
    - Better to require rising edge to avoid accidental start if button already held from previous action.
    - Possibly code resets portionButton usage somewhere.
    - We'll refine:
    - DosingMgr monitors portionButton. Could use a memory to ensure starting portion on press events, not just if held.
  + But given snippet, maybe simpler:
    - If portion mode active and portion not currently running and all conditions true at a check, then start pumping (dosingActive=TRUE).
    - Then quickly portionButton will likely be depressed (since it's momentary), but once pumping started, not reevaluating canStart each cycle (maybe gating, but dosingActive now true so they might skip evaluation).
    - They might incorporate an edge by requiring portionButton to be pressed each time from off state.
    - Possibly they handle by setting portionButton to 0 once used or requiring it to go false then true again.
  + Implementation for clarity:
    - maintain a flag portionInProgress (dosingActive), and portionCompleted (or portionDone already output).
    - If dosingActive=FALSE and awaitTrigger=FALSE at mode start:
    - If UsePortionButton=FALSE -> proceed to start portion immediately (like auto-run).
    - If UsePortionButton=TRUE -> do nothing (await button).
    - Meanwhile, monitor triggers:
    - If UsePortionButton=TRUE:
      * On rising edge of PortionButton OR NextDose:
      * if gun sensor condition is ok (if required), then begin portion (dosingActive=TRUE).
      * If sensor not ok, ignore until it is.
    - If UsePortionButton=FALSE:
      * if not dosingActive currently (and portion done or first portion), start new portion immediately (no delay).
      * Possibly they still ensure gun sensor is in place before each portion (makes sense).
    - Once portion started:
    - Reset SoftFlow (Volume=0),
    - FillPump\_ON = TRUE,
    - SoftFlow.Enable = TRUE.
    - Mark dosingActive=TRUE.
    - PortionDone flag remains FALSE for now.
    - During portion:
    - Monitor GunSensor:
      * If GunSensor lost (and required), abort portion:
      * FillPump\_OFF,
      * SoftFlow.Enable=FALSE,
      * dosingActive=FALSE,
      * do not set PortionDone (since incomplete).
      * Could wait for gun returned and maybe allow same portion resume? Or treat as aborted portion lost volume (semi-lost).
      * Simpler: user will likely scrap that portion, and possibly hit NextDose to try again.
      * We'll consider portion aborted and require next trigger as if portion done (with no done flag).
      * Might also raise an alarm or not? Possibly not needed, user will see pump stop.
    - Otherwise, monitor Volume:
      * If Volume >= SP\_DoseVolume:
      * FillPump\_OFF (pump stops exactly at portion volume or slightly after since no actual feedback loop, but accept slight overshoot).
      * SoftFlow.Enable=FALSE (stop counting, though it's basically done).
      * dosingActive=FALSE (portion ended).
      * Set PortionDone = TRUE for e.g. 1 cycle (maybe hold until reset next portion? But better as impulse).
        + Possibly code toggles it for one scan or a short timer to ensure HMI sees.
        + Many do: PortionDone true until next portion start, but requirement says "импульс".
        + We'll specify: set TRUE for e.g. 1 second or one PLC cycle (which might be too short for HMI to catch if 50ms, maybe keep it true until external ack? But they call it impulse, maybe a second or until Next portion).
        + But we have CMD\_NextDose separate, so likely they intended portionDone to act as a pulse output to external filler or just indicator.
        + We'll propose: portionDone remains TRUE for e.g. 1 second after pump stops, then resets automatically (or resets on next portion start).
      * If UsePortionButton=FALSE:
        + Immediately or after a short pause, begin next portion automatically:
        + Actually, might want a small pause to allow filling line to settle, or immediate.
        + Danger if portion on portion quick with no break? Possibly fine if continuous needed.
        + But maybe safer to have like a second or operator can always stop.
        + Perhaps no explicit pause coded; they might start next portion next scan:

Their code likely does: after portion done, still EN\_Portion true, no portion active, portionButton param false, so on next cycle canStart becomes true automatically -> start pump again if all conditions satisfied.

So yes, it might immediately restart pump (which might even cause portionDone impulse to be not easily observed if short).

Ideally, they'd incorporate slight delay or wait for portionDone ack.

CIP has commit delay concept, but here not.

So likely it will immediate loop.

* + - * + We'll state default: automatic continuous dosing until turned off.
        + If at some point operator toggles Mode\_DispDose off, that stops sequence.
      * If UsePortionButton=TRUE:
        + After portion done, DosingMgr sets awaitTrigger = TRUE ( meaning waiting for user input).
        + It will not auto restart.
        + It resets portionDone after maybe acknowledging or the next portion start.
    - Continue until out of portion mode.
* When EN\_Portion turned FALSE (mode ended):
  + If a portion was in progress, stop it (pump off).
  + Clear flags, maybe portion done not triggered if incomplete (or maybe still set abort? No, they likely skip done).
  + SoftFlow.Enable off.
  + PortionDone reset (likely ensure false).

**Code details guess from snippet [98†]:** - They set dosingActive := TRUE when conditions allow start. - They likely have separate logic to reset after volume done: - e.g. IF dosingActive THEN ... check volume ... IF volume >= SP THEN dosingActive := FALSE; PortionDone := TRUE; END\_IF; - And maybe outside: if dosingActive just turned false and still EN\_Portion, if UsePortionButton=FALSE then immediately re-evaluate (thus automatically next portion). - If UsePortionButton=TRUE, they might incorporate a latch requiring button press for next: - Possibly they set dosingActive false but don't allow it to become true again until canStart includes portionButton which by now is likely 0 then need next press.

* They may not explicitly use awaitTrigger in code, but logically it's integrated in the condition via portionButton.

Given potential complexity, we'll provide logic description:

**Логика DosingManager:**

* Когда режим порционного розлива активирован (EN\_Portion=TRUE):
* **Старт порции:**
  + Проверить условия: если PAR\_UsePortionButton=TRUE, ждать нажатия PortionButton или NextDose. Если PAR\_UsePortionButton=FALSE, не ждем (сразу можно лить).
  + Проверить датчик пистолета: если PAR\_UseGunSensor=TRUE, требовать GunSensor=TRUE (пистолет на месте).
  + Только при выполнении этих условий начать дозу:
  + Выставить FillPump\_ON = TRUE (насос включен).
  + Сбросить счетчик SoftFlowmeter (послать Reset=TRUE на один цикл).
  + Затем Enable = TRUE в SoftFlow (начать считать объем).
  + Установить внутренний флаг pouring = TRUE (идет налив).
  + Сбросить PortionDone (на случай, если с прошлого остался).
  + Если условия не выполнены – ждем. (Например, если нужен триггер, а кнопка не нажата – ничего не делаем; если пистолет не на месте – тоже).
* **Процесс дозирования (во время pumping):**
  + Постоянно мониторить:
  + Объем налитого: Volume = SoftFlow.VolumeTotal.
  + Если Volume >= SP\_DoseVolume:
    - Остановить насос: FillPump\_ON = FALSE.
    - Enable = FALSE в SoftFlow (хотя счетчик можно оставить, он остановится, т.к. PumpOn=FALSE).
    - Снять флаг pouring = FALSE (доза завершена).
    - Выдать сигнал PortionDone = TRUE (импульс). Можно удерживать TRUE, пока… (см. ниже).
    - Если PAR\_UsePortionButton=FALSE: сразу (на след. цикле) начинать новую порцию, *если режим всё еще активен*. (Алгоритм вернется на шаг 1, условия – GunSensor, etc. GunSensor скорее всего будет ок, так что мгновенно начнет).
    - Если PAR\_UsePortionButton=TRUE: не начинать новую. Установить флаг awaitingTrigger=TRUE (чтобы не стартовало до команды). То есть DosingMgr просто ждет теперь команды от пользователя.
  + Если GunSensor потерян (стал FALSE) во время наливания (и PAR\_UseGunSensor=TRUE):
    - Сразу остановить насос: FillPump\_ON=FALSE, Enable=FALSE.
    - pouring=FALSE.
    - Не устанавливать PortionDone (порция не полная). Можно занулить счетчик или оставить (по желанию, но лучше сбросить, т.к. остаток объема с предыдущей порции сбрасывать).
    - Можно поставить awaitingTrigger=TRUE так, как будто порцию прервали, и надо нажать кнопку заново.
    - Практически, оператор увидит остановку, возможно, вернет пистолет на место, и нажмет кнопку NextDose снова, чтобы дозалить остаток (или новую порцию).
    - ALM не делаем (не авария, просто условие).
  + (Если PortionButton pressed again mid-pour – можно игнорировать до завершения текущей, или если portion done triggered, any further press triggers next portion).
  + DosingMgr может обновлять STAT\_MixerRunning? Нет, это про мешалку, irrelevant here.
* **Окончание порции:**
  + Если PortionDone=TRUE установлен, DosingMgr должен его сбросить через небольшой промежуток:
  + Можно держать TRUE 1–2 сек, либо до начала следующей порции. Чтобы HMI улавливала событие. В данном проекте, PortionDone – Ram bool non-retain, HMI может на индикатор или trigger script.
  + Можно, например, через Main сбрасывать (Main может копировать PortionDone в MB\_ParMask b16?), но они явно ввели ParMask.b16 PortionDone (см. ParMask bits – b16 named PortionDone).
  + Yes, ParMask in HMI had b16 labeled “PortionDone”【13†】.
  + Possibly, DosingMgr sets global PAR\_UsePortionButton or rather an alias? Actually, they listed "b16 PortionDone" in ParMask.
  + It's unusual to put output in ParMask, but maybe they reuse it to send this status to HMI because ParMask is R/W modbus. If ParMask is double word (32 bits) and HMI writes only lower bits for params, they could use bit16 as feedback:
  + If so, DosingMgr could set ParMask bit16, and then maybe Main picks that up and resets after a moment.
  + But easier: We have separate global Dosing\_PortionDone, which can be tied to something.
  + Possibly, HMI configured to monitor Dosing\_PortionDone directly as tag.
  + We'll treat Dosing\_PortionDone as independent indicator on HMI (maybe bit lamp).
  + Implementation: We'll set PortionDone=TRUE for e.g. 1 sec, then DosingMgr resets to FALSE.
  + Or, since PLC cycle is short, better to hold a few cycles (like 20 cycles ~1 sec).
  + Could have a small timer or counter to hold it, or simply reset when the next portion starts to ensure HMI had time.
  + We'll say: portionDone resets either on next portion start or after X cycles, whichever earlier.
  + If UsePortionButton=FALSE (auto mode) – as noted, DosingMgr likely immediately loops to next portion since RunReq still true:
  + This means portionDone might flicker so fast the next portion resets it quickly. Might be okay if HMI reads quickly, but risk missing.
  + Possibly, they intended portionDone mostly for usePortionButton case or external integration (like send pulse to filler machine).
  + In auto mode, portionDone might not be needed or only final portion (if mode turned off) will yield a final done pulse perhaps.
  + We'll not complicate, just keep logic uniform.
* **Повтор порций/завершение режима:**
  + Если режим остаётся активным (EN\_Portion=TRUE):
  + Если PAR\_UsePortionButton=FALSE: цикл шаг1-3 повторяется немедленно.
  + Если PAR\_UsePortionButton=TRUE: цикл остановится на шаге 3 ждя нажатия. Когда оператор снова нажмет кнопку (PortionButton=TRUE) и GunSensor ok – DosingMgr повторит шаг1: FillPump\_ON=TRUE ...
  + Realize: they need to require the button to go to 0 and back to 1 (so likely they want rising edge).
  + So possibly internal logic ensures that after a portion is done, it won't restart portion until PortionButton was released and pressed again:
    - They may do: they checked canStart each scan. If after finishing portion, portionButton still True (like if held down), one might not want to start immediately (maybe wait for release).
    - But snippet doesn't show explicit edge detection, they may rely on physical button being momentary, user presumably releases it.
    - We'll assume typical usage: each portion requires one press, user won't hold it.
  + Если оператор завершает режим (EN\_Portion->FALSE via HMI Stop):
  + Если насос в это время лил (pouring=TRUE) – DosingMgr немедленно останавливает насос (FillPump\_OFF) mid-portion.
  + Не устанавливает PortionDone (порция не полная).
  + Сбрасывает internal states.
  + System then idle.
  + If user restarts mode, portion count resets.

**Инварианты Dosing:** - Single writer: only DosingMgr sets FillPump\_ON, others do not (Actuator uses that). - GunSensor enforcement: if enabled, pump never on when sensor false (ensured by logic at start and monitoring). - Volume accuracy: depends on correct PumpFlowRate and SoftFlow timing, no direct sensor, so moderate accuracy. - Multi-writing potential: None, DosingMgr solely controls FillPump. - SoftFlow meter integration: DosingMgr resets integrator at portion start to measure fresh.

### 2.7 FB\_VFDManager – управление частотным приводом мешалки

**Назначение:** Обеспечивает связь и управление частотным преобразователем (ПЧ) для двигателя мешалки. Если параметр PAR\_UseVFD=TRUE, VFDManager получает команду “Пуск/Стоп/Реверс” мешалки, а также заданную скорость от HMI, и формирует Modbus-запросы: командное слово (Start/Stop/Dir/Reset) и слово задания частоты. Также отслеживает статус ПЧ (слово статуса, бит аварии) и выдает сигнал аварии привода STAT\_VFD\_Fault. VFDManager разгружает основную логику от деталей Modbus-протокола и реализует heartbeat/контроль связи с приводом.

**Интерфейс:**

Inputs: | Signal | Dir | Type | Source (Main/Actuator) | Description | Default | |---------------------|-----|------|----------------------------|------------------------------------------------------------------|---------| | **PAR\_UseVFD** | In | BOOL | FB\_Main (ParMask b1)【56†】| 1 = использовать ПЧ для мешалки, 0 = локальное/аналоговое управление. VFDManager активен только если TRUE. | FALSE | | **MB\_SetMixerFreq** | In | UDINT| HMI (reg.518)【56†】 | Заданная скорость мешалки, в шкале 0..1000 = 0..100.0% от 50 Гц. (0 = 0%, 1000 = 100%). Задает желаемую частоту вращения. | 0 | | **MixerRunCmd** | In | BOOL | ActuatorMgr/Main | Команда “Пуск вперед” для мешалки. Источники: в авто Pasteur/CIP – ActuatorMgr при VFD=TRUE не включает DO3, но должен дать сигнал сюда. Вручную – Main при Mode\_Manual\_ACTIVE&UseVFD. | FALSE | | **MixerRevCmd** | In | BOOL | ActuatorMgr/Main | Команда “Пуск реверс” для мешалки. (Если реализуется, у нас ManualRev bit). | FALSE | | **MixerStopCmd** | In | BOOL | ActuatorMgr/Main | Команда “Стоп” мешалки. Может формироваться автоматически при переходах режимов (например, CIPStop) либо вручную оператором (Manual off). | FALSE |

Outputs: | Signal | Dir | Type | Destination | Description | Default | |-------------------|-----|------|---------------------------|--------------------------------------------------------------|---------| | **VFD\_SetFreqWord** | Out | UDINT | Modbus (Slot2 reg.4096 or 36864)【18†】 | Выходное слово задания частоты ПЧ. Формируется на основе MB\_SetMixerFreq (0..1000 -> 0..50000, т.е. в сотых Гц). Например, MB\_Set=500 => SetFreqWord=25000 (25.00 Гц). Записывается функцией 0x06 или 0x10 в регистр 0x1000 (или 0x9000). | 0 | | **VFD\_CmdWord** | Out | UDINT | Modbus (Slot2 reg.8192)【18†】 | Командное слово ПЧ. Формат: бит0 RUN/FWD, бит1 RUN/REV, бит2… (AD30: 1=FWD,2=REV, 5=CoastStop,6=DecelStop,7=Reset)【15†】. VFDManager выставляет соотв. код: при MixerRunCmd -> 1, MixerRevCmd->2, MixerStopCmd->6, а также 7 (Reset) если надо сбросить. | 0 | | **VFD\_Status** | Out | UDINT | Global (read by Main)【57†】 | Слово статуса ПЧ (копия прочитанного регистра 0x3000). Interpreted bits: e.g., 1=FWD running, 2=REV running, 3=STOP. Обновляется по данным Modbus чтения (в OwenLogic, либо Main читает регистр и пишет сюда). | 0 | | **VFD\_Fault** | Out | BOOL | FB\_Main (STAT\_VFD\_Fault) | Флаг аварии привода. Устанавливается, если статус ПЧ сообщает ошибку (в AD30, возможно, если в статусе определенный бит). Если AD30 выдаёт код >3 (например, 4=Fault?), надо уточнить. Предположим: VFD\_Fault=TRUE, если привод в аварии. Этот сигнал Main копирует на ALM\_MixerFault или отдельно. | FALSE |

Internals: - CommOK (BOOL) – может следить, ответил ли ПЧ на последние запросы. - heartbeatCounter – можно посылать периодически toggle bit in CmdWord? Not needed likely, PCH doesn’t require heartbeat, but maybe drive trips if no command after x time? Possibly no, AD30 might stop after comm loss if configured. - We'll focus on simpler logic:

**Логика VFDManager:**

* **Команды формирования:**
* При каждом цикле, VFDManager смотрит входы:
  + Если PAR\_UseVFD=FALSE: можно сразу выходы = 0, не делать ничего (выходы AO2/DO3 handle analog).
  + Если PAR\_UseVFD=TRUE:
  + Рассчитывает VFD\_SetFreqWord:
    - freq\_pct = MB\_SetMixerFreq (0..1000).
    - maxFreq = 50.00 Hz (P0-14).
    - VFD\_SetFreqWord = (freq\_pct \* 50.00), но с масштаб 0.01Hz:
    - Actually, better: 0..1000 corresponds 0..100%, so
    - if using reg 0x1000: need -10000..10000 for -100..100%. They likely decided to use 0x1000 (4096).
    - If so, to set forward direction, they'd put positive values. However, if using CmdWord for direction, maybe they'd rather use 0x9000 (which uses sign bit in value? Actually 0x9000 was 0..50.00 Hz).
    - According to CSV: they listed both 0x1000 (with range -10000..10000) and 0x9000 (with 0..P0-14).
    - Possibly they intended to use 0x9000 (register 36864) for clarity (0..5000 for 0..50.00 Hz).
    - But in code they set VFD\_SetFreqWord in UDINT expecting 0..50000 (makes me think 0x9000).
    - Actually, in code snippet【57†】: "VFD\_SetFreqWord: UDINT 0..50000 = 0..50.00 Hz (step 0.01 Hz)" – indicates they picked 0x9000 (where 50000 corresponds to 50.00 Hz).
    - So they likely write to reg 36864.
    - Thus, formula: VFD\_SetFreqWord = MB\_SetMixerFreq \* 0.05 (because MB\_SetMixerFreq is 0..1000 meaning 0..100.0%; 100% corresponds to 50.00 Hz which is 5000 in 0.01 increments).
    - Actually, if MB\_SetMixerFreq=1000 => we want 5000 (for 50 Hz).
    - So indeed: SetFreqWord = MB\_SetMixerFreq \* 5.
    - E.g., MB\_Set=500 -> SetFreqWord=2500 (25.00 Hz).
    - But code says 0..50000, maybe MB\_Set was 0..1000 as decimal, multiply by 50? No, that yields 50000 if 1000. Actually, if MB\_Set 1000, \*50 = 50000 (which matches 0..50000).
    - Wait, 50000 at reg means 500.00 Hz if step 0.01? AD30 likely cannot 500 Hz. Possibly 0..50000 means units of 0.001? Or maybe P0-14 could be 500 Hz? Unlikely, AC drives rarely go that high.
    - There's confusion: AD30 likely can be parameterized with base freq maybe 50 or 60, rarely 500. Possibly P0-14 (max freq) could be 50 or 100, but not 500.
    - Possibly slight mismatch: or they set scale incorrectly. But I'll trust comment: 0..50000 for 0..50.00 Hz, likely meaning 1 unit=0.001 Hz (no, 50000 units for 50.00 means 1=0.001 indeed).
    - Actually 50000 corresponds 50.000 if 3 decimals, so maybe AD30 registers have 3 decimal places? But CSV says step 0.01, which would be 5000 for 50.00.
    - Possibly misprint, maybe it's 0..5000 (should be).
    - I'd assume 0..5000, but they wrote 50000, maybe AD30 uses 2 decimal points differently? Could be a specific of 0x9000 region.
    - Not essential to dive deeper, proceed conceptually.
  + Формирует VFD\_CmdWord:
    - If MixerRunCmd TRUE -> CmdWord = 1 (FWD).
    - Else if MixerRevCmd TRUE -> CmdWord = 2 (REV).
    - If both FALSE (and previously running) -> send Stop command:
    - Possibly differentiate Coast vs Decel stop:
      * They might use decel stop normally (6).
      * If emergency, maybe use coast (5).
      * But not sure how they'd choose. Possibly always decel (6) for normal stop.
    - If fault occurred and cleared, they might send Reset (7) after a fault (One could manage an internal state: if Fault just became active or after user acknowledges, send 7).
      * Hard to guess if implemented. Possibly not in initial scope.
    - Possibly handle that externally, like if ALM\_MixerFault cleared, then send Reset.
    - Keep it minimal: If an edge in an internal needReset var, send 7.
  + If no command (stop) then maybe send 0 or 6:
    - AD30 doc: 0 might do nothing (if drive not running then fine), might have to explicitly send stop command one time.
    - Possibly they do:
    - if run->stop edge: send 6 once, then revert to 0 or maintain 6? But if keep sending 6 repeatedly, might not matter or might matter.
    - Simpler: always if no run bits true, send code 6 (decel stop) continuously.
    - Or maintain last command until changed? But then drive might re-run if power cycled?
    - We guess they send Stop continuously when not running.
  + Also might incorporate heartbeating:
    - Not needed, but sometimes one toggles a "Run Enable" bit in command word each cycle to show comm alive. But AD30 likely doesn't need that.
  + Combined logic:
  + cmd = 0;  
    if (MixerRunCmd) cmd = 1;  
    else if (MixerRevCmd) cmd = 2;  
    else if (prevRun || MixerStopCmd) cmd = 6;  
    // If fault condition and maybe user triggered reset, cmd=7 for one cycle.  
    VFD\_CmdWord = cmd;
* **Статус чтение:**
  + VFD\_Status input presumably updated by reading reg 0x3000 periodically (maybe done by OwenLogic comm config).
  + If not done automatically, VFDMgr could include reading in its logic, but likely done by adding remote variables (which CSV suggests).
  + Actually CSV had lines for monitoring:
  + They might have configured those as separate variable triggers by polling tasks.
  + Possibly, one could assume VFD\_Status (UDINT) is tied to remote read of 0x3000, updated background.
  + If not, VFDMgr might call a function to read it but not visible in ST likely. Instead, they might rely on the remote variable concept:
  + VFD\_Status as remote with read func 0x03, which means system polls it.
  + So presumably, VFD\_Status out updated asynchronously (but in code treat as input?), but they listed as output of block that is updated externally is odd.
  + Could be they plan to read it in Main and feed VFD\_Status to VFDMgr via Var\_Input... But they put it in Var\_Output in code【57†】 with comment "updated externally".
  + So likely VFD\_Status is a global updated by comm driver, but made accessible via FB output for completeness.
  + We'll not fixate – just say VFDMgr yields status and fault.
  + VFD\_Fault:
  + Determine from VFD\_Status:
    - If status code indicates fault or if no response:
    - AD30 likely has separate fault register (maybe 0x3001 or something).
    - But none given; maybe status 3 means stopped, 4 means fault? or there's a bit.
    - In absence, simple: If cannot reach drive or if drive internal fault flagged, set STAT\_VFD\_Fault True.
    - Possibly they planned to parse an error code but skipped detail.
  + COMM check:
  + VFDMgr or Main could track if modbus read fails (some systems have Comm status bits).
  + They had STAT\_VFD\_Online but not used in VFDMgr code snippet.
  + Possibly, Main uses MB\_CommFlags (reg.562) to hold comm status (maybe bit for VFD link).
  + We have MB\_CommFlags at reg.562 in slot1 CSV:
    - Possibly bit0 = HMI link, bit1 = VFD link, etc.
    - If so, Main could decode MB\_CommFlags from slot2 or from a internal, not sure.
    - It's not clearly done. Instead they had STAT\_VFD\_Online (Ram).
    - Could be updated by comm logic or user code.
  + Suggest:
    - If VFD doesn't respond to a read within, say, 3 seconds, set STAT\_VFD\_Online=FALSE.
    - We incorporate in Main: if VFD\_Status hasn't updated or some error code from comm, then offline.
    - For TЗ, mention needed and what safe state (stop mixer, alarm).
  + Reaction: Already in Actuator, if PAR\_UseVFD and lost comm, ALM\_MixerFault (or separate) should trigger (bit4 presumably).
  + Implementation: if offline and Mixer supposed to run, likely no command gets through, but mechanical might be off anyway. But better safe: if offline, treat as if stop command not guaranteed, so set an alarm and ensure no run commands being sent.
* **Manual control integration:**
* When ManualActive and UseVFD=TRUE, ActuatorMgr sets DO3/DO4 off but doesn't directly command VFD. We rely on:
  + Main should convert manual bits to VFD Manager inputs:
  + e.g., if ManualActive:
    - if Man3 (FWD) pressed -> MixerRunCmd=TRUE.
    - if Man4 (REV) pressed -> MixerRevCmd=TRUE.
    - if none pressed or previously running and now off -> MixerStopCmd=TRUE.
  + Possibly, Main can do that logic or Actuator could have done via VFD\_MixerRunReq input, but Actuator didn't implement it.
  + Actually, Actuator had VFD\_MixerRunReq input but not used in code. Possibly they intended to set VFD\_MixerRunReq = Man3, VFD\_MixerRevReq = Man4 in Actuator (like inside manual branch, if PAR\_UseVFD then maybe treat those differently).
  + Simpler: do it in Main: if Mode\_ManualActive and PAR\_UseVFD:
    - Use the manual bits to control VFDMgr.
  + Considering time, we specify it qualitatively:
    - Manual: pressing Man3 or Man4 will cause VFD run fwd or rev. Releasing them -> stops.

**Инварианты:** - When PAR\_UseVFD=TRUE: - DO3/DO4 should remain off (Actuator ensures this). - AO2 not used (0). - Mixer controlled solely via VFD commands from VFDMgr. - Single writer: - Only VFDMgr writes VFD\_CmdWord and VFD\_SetFreqWord (no conflict). - Only VFDMgr (or comm driver) sets VFD\_Status and VFD\_Fault. - Safety: - Loss of comm or drive fault triggers stoppage of mixer and ALM\_MixerFault.

### 2.8 FB\_SoftFlowmeter – программный расходомер

**Назначение:** Расчет объема продукта, перекаченного насосом розлива, на основе заданной производительности и времени работы насоса. Эмулирует показания расходомера для дозатора при отсутствии физического счетчика. Обнуляется по команде и интегрирует объем, пока насос включен. Служит для порционного розлива (DosingManager) – определяет момент достижения требуемого объема.

**Принцип работы:** Если насос включен (PumpOn=TRUE), SoftFlowmeter каждую секунду добавляет к накопленному объему Rate\_Ls литров в секунду. Rate\_Ls – заданная производительность (л/с), предполагается постоянной. Используя секундный тик Seconds\_UD, блок отслеживает реальное время, чтобы не зависеть от цикла ПЛК.

**Интерфейс:**

Inputs: | Signal | Dir | Type | Source (Main/Dosing) | Description | Default | |----------------|-----|------|---------------------------|------------------------------------------------------------|---------| | **Enable** | In | BOOL | DosingManager.FillPump\_ON【84†】 | Разрешение интеграции. TRUE = начать считать объем (когда насос работает). Обычно подключается прямо к сигналу насоса. | FALSE | | **Seconds\_UD** | In | UDINT| FB\_Main.Seconds\_UD【61†】 | Секундный монотонный счетчик (каждая единица = 1 сек). Используется для измерения прошедшего времени. | 0 | | **PumpOn** | In | BOOL | DosingManager.FillPump\_ON【84†】 | Сигнал фактической работы насоса (можно дублировать Enable). Если PumpOn=FALSE, счет объема останавливается. | FALSE | | **Rate\_Ls** | In | REAL | FB\_Main.PAR\_PumpFlowRate | Производительность насоса, л/с (на номинальной скорости). Предполагается постоянной. | 0.0 | | **Reset** | In | BOOL | DosingManager (Portion start)【91†】 | Сигнал сброса накопленного объема. При TRUE, SoftFlowmeter обнуляет счетчик VolumeTotal. Дается импульсом (на один цикл). | FALSE |

Outputs: | Signal | Dir | Type | Consumers | Description | Default | |----------------|-----|------|---------------------|------------------------------------------------------------|---------| | **VolumeTotal**| Out | REAL | DosingManager.FlowMeter (internal) | Накопленный объем с момента последнего сброса, в литрах. Обновляется, когда Enable=TRUE. DosingManager сравнивает с SP\_DoseVolume. | 0.0 | | *(Optionally could output instantaneous flow or something, but not required.)* |

**Логика SoftFlowmeter:**

* Когда Reset=TRUE, SoftFlowmeter устанавливает VolumeTotal = 0. (Можно делать при фронте Reset).
* Идея: Seconds\_UD – глобальный секундный счетчик. SoftFlowmeter может запоминать lastSec (UDINT).
* Если Enable=TRUE (интегрируем), SoftFlowmeter на каждом цикле:
* Смотрит текущий Seconds\_UD. Если Seconds\_UD != lastSec (значит прошла 1 сек),
* Вычисляет deltaSec = (Seconds\_UD - lastSec). Если пропущено несколько секунд (маловероятно, но если задержка), интегрировать за все.
* Добавляет Rate\_Ls \* deltaSec к VolumeTotal.
* Обновляет lastSec = current.
* Если Enable=FALSE (насос не работает), SoftFlowmeter не увеличивает VolumeTotal и, возможно, обновляет lastSec (чтобы не интегрировать паузу).
* Так обеспечивается нечувствительность к частоте цикла (только к точности Seconds\_UD).
* Alternatively, simpler: use PLC cycle time: VolumeTotal += Rate\_Ls \* cycleTime (s). But Seconds\_UD approach is fine if exact seconds. Might cause quantization error if lastSec missed partial second at start/stop, but negligible for large volumes or pump flows.
* Possibly, do partial second integration:
* e.g., if Enable turned true in between second ticks, it won't count until next sec tick changes. This might lose up to <1 sec volume. Acceptable given volumes likely >>1s pumping.
* Could refine by also adding fraction if e.g. just turned on, but complexity may not needed for coarse measurement.
* After VolumeTotal updated, SoftFlowmeter sets output.
* If Reset came at same time as pump on, the logic resets first then counts new.
* SoftFlowmeter doesn’t directly control anything, it’s passive after being triggered by Dosing.

**Инварианты:** - VolumeTotal only increases when pump on. - If Rate\_Ls not calibrated, volume calc can be off; calibrate Rate for near correct volume. - Single writer: SoftFlow sets VolumeTotal, Dosing reads it, no conflict. - Reset by Dosing ensures no carryover between portions. - If pump stops mid-portion (gun removed or stop), VolumeTotal holds partial volume delivered (which one might subtract from portion target if portion aborted? Possibly the portion is incomplete and you cannot 'undo' that product; presumably they scrap or manual measure additional). - If portion aborted, likely they will either discard or manually complete it. System may either reset Volume or keep it. Possibly better to reset on abort so that next portion starts fresh (maybe losing track of spilled volume, but that spilled volume isn't delivered to target container anyway). - We'll not dive deeper.

**Взаимодействие DosingManager – SoftFlowmeter:** - At portion start, Dosing sends Reset=TRUE one cycle, then sets Enable=TRUE (pump on). - SoftFlow resets volume then starts counting. - Dosing monitors VolumeTotal; when target reached, it stops pump (and may or may not send Reset immediately for next portion? It likely resets only at next portion start because if multiple portions without leaving mode, we want to accumulate only within portion). - Actually, they likely reset at each portion start, not after each portion done (or it wouldn't measure the portion, oh actually you have to reset at start of portion). - So yes, Dosing resets at start, not at end. - So after portion done, VolumeTotal holds ~ portion size (slightly above maybe). - They might leave it until next portion start resets it again. - Or in continuous mode, might never reset unless user press ResetFlow (there is CMD\_ResetSoftFlow (Mask1.b15) which could be an external command to reset volume count). - Yes, user can press "Reset Flowmeter" (b15) to zero the count manually (maybe to track total dispensed in continuous mode). - We have CMD\_RestSoftFlow (likely Reset SoftFlow, b15). - In code, DosingManager or SoftFlow should respond to that too: - Possibly Main sets global CMD\_ResetSoftFlow on bit detection. - DosingMgr could then trigger SoftFlow.Reset if not currently doing portion (like continuous). - But simpler: if continuous mode, user might want to zero the counter at some point; if we always reset at portion start in dose mode, continuous mode never resets unless user triggers. - So implement: - if CMD\_ResetSoftFlow goes true, SoftFlowmeter.Reset=TRUE for a cycle no matter the mode. - Actually, easier: Main or Dosing could link CMD\_ResetSoftFlow to SoftFlow.Reset. - We have global CMD\_RestSoftFlow (looks like a typo missing e in "Rest"? likely Reset). - Possibly Main or Dosing does: if CMD\_ResetSoftFlow then SoftFlow.Reset=TRUE. - We'll note that as feature.

**Паспорта блоков завершены.** Далее:

## 3. Словарь сигналов (общие переменные)

Этот раздел содержит сводную таблицу всех глобальных переменных проекта – их назначение, тип, единицы, диапазоны, область видимости и “единственный писатель” (какой блок отвечает за обновление). Полный машинно-читаемый словарь приведен в приложении **Signals\_Dictionary.csv**, здесь приведем основные категории.

Для удобства, разделим сигналы на группы:

**3.1 Дискретные входы (DI) и выходы (DO) контроллера:**

| Signal | Type | I/O | HW Addr | Source/Writer | Consumers | Description & Semantics | Default | Notes |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DI1\_MixerFault** | BOOL | DI | X1.1 | HW (NO contact, NЗ circuit) | Main, ActuatorMgr | Авария привода мешалки (концевой контакт схемы ПЧ/пускателя). 1 = Норма, 0 = Сработало аварийное реле. При 0: ALM\_MixerFault=TRUE, блокировка мешалки. | – | НЗ контакт, поэтому 0=авария. |
| **DI2\_HeaterFault** | BOOL | DI | X1.2 | HW (NO contact, NЗ) | Main, ActuatorMgr | Авария нагрева (термостат ТЭНов). 1 = Норма, 0 = Перегрев/отключение нагр. При 0: ALM\_HeaterFault=TRUE, BLOCK\_HeaterOff=TRUE. | – | НЗ цепь. |
| **DI3\_LevelJacket** | BOOL | DI | X1.3 | HW (NO contact, NЗ) | Main, ActuatorMgr | Датчик уровня в рубашке (наличие воды). 1 = Уровень есть (норма), 0 = уровень пропал. При 0: ALM\_NoWaterJacket=TRUE, BLOCK\_HeaterOff=TRUE. | – | НЗ датчик уровня. |
| **DI4\_PortionBtn** | BOOL | DI | X1.4 | Кнопка (NO) | DosingManager | Кнопка “пуск порции”. 1 = нажата. Используется в порционном режиме, если PAR\_UsePortionButton=1, для запуска каждой порции. | 0 | Кратковременная кнопка. |
| **DI5\_GunSensor** | BOOL | DI | X1.5 | Датчик положения “пистолета” (NO) | DosingManager | 1 = пистолет **снят** (контакт разомкнут), 0 = на месте (контакт замкнут). При PAR\_UseGunSensor=1, насос розлива работает только когда датчик = 0. | 0 | Подключен как НО (даёт сигнал, когда пистолет снят). |
| **DI6\_ProductFlow** | BOOL | DI | X1.6 | Датчик потока (NO) | CIPManager (ALM) | 1 = поток/жидкость присутствует в контуре (например, проточный датчик). 0 = нет потока. Используется CIPManager для контроля наличия моющей жидкости: при 0 -> ALM\_NoWater (отсутствует среда). | 0 | Если нет физ. датчика, можно перемычкой держать 1. |
| **DI7\_EStop** | BOOL | DI | X1.7 | Грибок “Стоп” (NC) | Main, ActuatorMgr | 1 = норма (грибок не нажат), 0 = аварийный стоп нажат. При 0: BLOCK\_AllOff=TRUE (все выходы отключаются немедленно), ALM\_EStop (если предусмотреть). | 1 | НЗ кнопка авар.стоп. |

| **DO1\_ValveCold** | BOOL | DO | Y1.1 | ActuatorManager【42†】 | Клапан холодной воды рубашки (охлаждение пастеризатора). 1 = открыт (подача ХВС в рубашку). Управляется PasteurManager (охлаждение) или вручную Man1. | 0 | Электромагн. клапан или соленоид. | | **DO2\_CircPump** | BOOL | DO | Y1.2 | ActuatorManager【43†】 | Циркуляционный насос контура рубашки. 1 = включен (гонит воду/гликоль). Включается при пастеризации (нагрев и охлаждение) и вручную Man2. | 0 | Может отсутствовать, если рубашка проточная – тогда всегда 0. | | **DO3\_MixerFwd** | BOOL | DO | Y1.3 | ActuatorManager【107†】 | Привод мешалки – вращение вперёд. 1 = мешалка вращается (прямое направление). В авто: включается при Pasteur/CIPActive (если надо мешать). В Manual: по Man3. | 0 | Если PAR\_UseVFD=TRUE, этот DO не используется (ПЧ вместо него). | | **DO4\_MixerRev** | BOOL | DO | Y1.4 | ActuatorManager【107†】 | Привод мешалки – реверс. 1 = вращение в обратную сторону. Используется только вручную (Man4), автоматические режимы держат 0 (кроме, возможно, спец. рецепт). | 0 | Если мешалка конструктивно не реверсируется, этот DO можно не подключать. | | **DO5\_PumpDisp** | BOOL | DO | Y1.5 | ActuatorManager【86†】 | Насос розлива продукта. 1 = включен, подает продукт на разлив. Управляется DosingManager (DispenseActive) или вручную Man5. | 0 | Насос типа центробежный или объемный, подключен через пускатель. | | **DO6\_Heater1** | BOOL | DO | Y1.6 | ActuatorManager【40†】【118†】 | Нагреватель группы 1 (или электроклапан подачи пара). 1 = включен. Управляется TemperatureManager (DO\_Heater1\_Cmd) с учетом блокировок, либо вручную Man6. | 0 | Если PAR\_UseSteam=1, DO6 включается при запросе нагрева (открыть пар). | | **DO7\_Heater2** | BOOL | DO | Y1.7 | ActuatorManager | Нагреватель группы 2. 1 = включен. Учитывается PAR\_HeaterGroup2Enabled. Вручное управление Man7. | 0 | При PAR\_HeaterGroup2Enabled=0 всегда 0 (кроме ручного). | | **DO8\_Heater3** | BOOL | DO | Y1.8 | ActuatorManager | Нагреватель группы 3. 1 = включен. Учитывается PAR\_HeaterGroup3Enabled. Man8 – вручную. | 0 | – | | **DO9\_ValveCIP\_Cold**| BOOL | DO | Y2.1 | ActuatorManager【107†】 | Клапан подачи холодной воды для CIP. 1 = открыт подача ХВС в систему CIP. Управляется CIPManager (во flush этапах) или вручную Man9. | 0 | – | | **DO10\_ValveCIP\_Hot**| BOOL | DO | Y2.2 | ActuatorManager | Клапан горячей воды CIP. 1 = открыт подача ГВС. CIPManager или Man10. | 0 | – | | **DO11\_ValveAlk** | BOOL | DO | Y2.3 | ActuatorManager | Клапан подачи щелочного раствора CIP. 1 = открыт. CIPManager (циклы щелочи) или Man11. | 0 | – | | **DO12\_ValveAcid** | BOOL | DO | Y2.4 | ActuatorManager | Клапан подачи кислотного раствора CIP. 1 = открыт. CIPManager (циклы кислоты) или Man12. | 0 | – | | **DO13\_PumpCIP** | BOOL | DO | Y2.5 | ActuatorManager | Насос CIP. 1 = работает (циркуляция моющего раствора). CIPManager включает на этапах, вручную Man13. | 0 | – | | **DO14\_ValveDrain** | BOOL | DO | Y2.6 | ActuatorManager | Клапан слив CIP / зуммер. 1 = открыт слив моющего раствора в дренаж (или включен звуковой сигнал). CIPManager на этапах слива (и Manual Man14). | 0 | На выход DO14 может висеть зуммер, тогда CIPDrainValve физически не используется – [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ конфигурации]. |

**3.2 Аналоговые входы (AI) и выходы (AO):**

| Signal | Type | I/O | HW Addr | Source/Writer | Consumers | Description | Scale/Units | Default |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **AI\_TempProduct** | UDINT | AI | AI1 | HW (NTC sensor) | FB\_Main (conv)【68†】 | Температура продукта (датчик в емкости). Значение в формате ×10 (целочисленно). Например 250 -> 25.0 °C. Приводится к REAL в PV\_TempProduct\_Raw. | 0.1 °C/LSB | – |
| **AI\_TempJacket** | UDINT | AI | AI2 | HW (NTC sensor) | FB\_Main (conv) | Температура теплоносителя (рубашки). Формат ×10 °C (UDINT). Конвертируется в PV\_TempJacket\_Raw. | 0.1 °C/LSB | – |
| **AO\_HeaterPower** | REAL | AO | AO1 | ActuatorMgr (from TempMgr)【120†】 | Hardware (Analog Out 0–10V) | Аналоговый выход на систему нагрева. Интерпретация: 0 = 0%, 1000 = 100% (по CSV) – соответствует 0–10 В. Управляет либо клапаном пара (при паровом нагреве), либо может быть не задействован при электричестве (но м.б. подключен индикатор мощности). В TempMgr/PID: 0–100%. | 0–1000 (0–100%) | 0 |
| **AO\_MixerSpeed** | REAL | AO | AO2 | ActuatorMgr【89†】 | Hardware (Analog Out 0–10V) | Аналоговый сигнал управления скоростью мешалки (для частотника с аналоговым входом или аналогового регулирования). 0 = 0%, 1000 = 100%. При PAR\_UseVFD=FALSE, ActuatorMgr выставляет 100% когда мешалка включена, иначе 0. При PAR\_UseVFD=TRUE – всегда 0 (не используется). | 0–1000 (0–100%) | 0 |
| **AI\_… резерв** | – | – | – | – | – | Дополнительные аналоговые каналы не задействованы. | – | – |

**3.3 Сетевые переменные (Modbus, Slot1 – HMI):**

*(Адреса Modbus указаны в десятичном представлении регистров Holding)*

| Signal | Type | Reg | Direction | Source/Writer | Consumers | Semantics | Default |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **MB\_Alarms** | UINT | 512 | PLC→HMI | FB\_Main (ALM\_Mask\_UD) | HMI (индикатор аварий) | Маска аварийных сигналов (16 бит). См. раздел 3.4 ниже по битам. PLC обновляет на основании ALM\_\* флагов. | 0 |
| **MB\_OutStateMask** | UINT | 517 | PLC→HMI | FB\_Main (STAT\_OutStateMask)【127†】 | HMI (схема) | Маска состояний выходов DO (бит0 = DO1, ... бит13 = DO14). 1 = соответствующий выход включен. Отображается на мнемосхеме. | 0 |
| **MB\_SystemStatus** | UINT | 519 | PLC→HMI | FB\_Main | HMI (индикатор режимов) | Системный статус (битовая маска): b0=ManualActive, b1=AutoActive, b2=AlarmsPresent, b3=AlarmResetRequest (или сброс произошел). Формируется Main. | – |
| **MB\_TempProduct\_x10** | INT | 513 | PLC→HMI | FB\_Main (PV\_TempProduct\_Raw\*10?) | HMI (дисплей) | Температура продукта, 0.1 °C. PLC может напрямую копировать AI\_TempProduct (с учетом валидности) или брать PV\_TempProduct (фильтрованную) и умножать на 10. Рекомендуется отображать реальное значение (фильтрованное). | 0 |
| **MB\_TempJacket\_x10** | INT | 514 | PLC→HMI | FB\_Main | HMI (дисплей) | Температура рубашки, 0.1 °C. Аналогично продуктовой. | 0 |
| **MB\_CIPStagesMask** | UINT | 515 | PLC→HMI | CIPManager【121†】 | HMI (индикатор) | Маска этапов CIP, которые будут выполняться в запущенном режиме. Например, Full=0x7F, Alk=0x67, Acid=0x73, Rinse=0x60. Устанавливается при старте CIP. | 0 |
| **MB\_PasteurStagesMask** | UINT | 516 | PLC→HMI | PasteurManager (calc) | Маска этапов пастеризации. Возможно, используется для отображения стадий (Heat/Hold/Cool bits). Например, 0x7 = все три. В текущей реализации, Pasteur всегда проходит все стадии, маска может не обновляться. [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ использования] | 0 |  |
| **STEP\_Pasteur** | INT | 545 | PLC→HMI | PasteurManager | HMI (этап пастеризации) | Код текущей стадии пастеризации. 0=Не запущена, 1=Нагрев, 2=Выдержка, 3=Охлаждение, 4=Завершено, 5=Прервано. Обновляется PasteurMgr. | 0 |
| **STEP\_CIP** | INT | 546 | PLC→HMI | CIPManager | HMI (этап CIP) | Текущий этап CIP (номер цикла 1–7, или 0 если нет CIP). | 0 |
| **STEP\_Dispense** | INT | 547 | PLC→HMI | DosingManager? (not explicit) | HMI (этап розлива) | Стадия процесса розлива: 0=Выключен, 1=Непрерывный активен, 2=Льется порция, 3=Порция завершена (ожидание), 4=Прервано. [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ] – может рассчитываться Main по состояниям DosingMgr. | 0 |
| **MB\_SetMixerFreq\_0\_1pct50** | UINT | 518 | HMI→PLC | Оператор (с панели) | VFDManager【56†】 | Заданная скорость мешалки, в 0.1% от 50 Гц. Диапазон 0–1000 (соответствует 0.0–100.0%). Задаётся вручную на HMI. Используется VFDManager (при UseVFD) или игнорируется (при аналоговом управл.). | 0 |
| **PAR\_MaxMixerSpeed\_0\_1pct50** | UINT | 536 | HMI→PLC (retain) | Оператор (параметр) | ActuatorMgr (может использовать) | Ограничение максимальной скорости мешалки, 0.1% от 50 Гц. По умолчанию 1000 (100%). М.б. применено ActuatorMgr для аналог. вывода (AO2) – пока не реализовано, но можно ограничивать SetFreq. [Актуально, если нужно ограничить скорость] | 1000 |
| **MB\_TimeModeDisplay** | UINT | 520 | PLC→HMI | FB\_Main (calc) | HMI (индикатор) | Режим отображения времени (кода) – возможно, показывает, какой таймер вывести на экран (напр., CIP time vs Pasteur time). [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ]. | 0 |
| **MB\_WashTimeMin** | UINT | 521 | PLC→HMI | CIPManager? | HMI (индикатор) | Оставшееся время мойки – минуты. CIPManager может рассчитывать на основании активных таймеров. [Возможно, суммарное оставшееся время CIP]. | 0 |
| **MB\_WashTimeSec** | UINT | 522 | PLC→HMI | CIPManager? | HMI (индикатор) | Оставшееся время мойки – секунды (модуло минуты). | 0 |
| **SP\_HeatTemp\_x10** | UINT | 523 | HMI→PLC (setpoint) | Оператор | PasteurManager.SP\_HeatTemp【76†】 | Уставка температуры пастеризации, 0.1°C. Диапазон напр. 500–950 (50.0–95.0°C) – контролируется Main. Main преобразует в REAL °C. | 0 |
| **SP\_HeatTime\_min** | UINT | 524 | HMI→PLC (setp) | Оператор | PasteurManager (HeatTimeout) | Лимит времени нагрева, мин. 0 = не задан (выкл). Если >0, PasteurMgr следит, чтобы нагрев достиг SP за это время, иначе STAT\_HeatTimeout. | 0 |
| **SP\_SmoothTemp\_x10** | UINT | 525 | HMI→PLC | (Резерв) | PasteurManager (not used) | “Температура выравнивания” – видимо, промежуточная температура плавного разгона. В текущей логике не задействовано. | 0 |
| **SP\_SmoothDiff\_x10** | UINT | 526 | HMI→PLC | PasteurMgr (cooling hysteresis)【79†】 | Дифференциал выравнивания, 0.1°C. Используется как гистерезис по температуре рубашки: при охлаждении пастеризации, клапан закрывается когда Tрубашки ≤ SP\_CoolTemp - Diff, и открывается обратно > SP\_CoolTemp - Diff【79†】. Рекоменд. 20 (2.0°C). | 20 (пример) |  |
| **SP\_HoldTime\_s** | UINT | 527 | HMI→PLC | Оператор | PasteurManager.SP\_HoldTime【76†】 | Время выдержки пастеризации, сек. Например 180s (3 мин). | 0 |
| **SP\_CoolTemp\_x10** | UINT | 568 | HMI→PLC | Оператор | PasteurManager (cooling target) | Уставка температуры охлаждения, 0.1°C. До какой температуры охлаждать продукт. 0 = не применять (значит не охлаждать активно?). Если >0, PasteurMgr охлаждает до этой T. Например, 300 = 30.0°C. | 0 |
| **MB\_CIP\_Edit\_T\_COLD** | UINT | 530 | HMI↔PLC | Оператор/HMI CIP screen【8†】 | CIPManager (edit) | Время подачи холодной воды (сек) для редактируемого этапа CIP. Оператор меняет, CIPMgr копирует в соответствующий CIP\_C#\_T\_COLD\_IN. | 0 |
| **MB\_CIP\_Edit\_T\_ALK** | UINT | 531 | HMI↔PLC | Оператор | CIPManager | Время подачи щелочи (сек) для выбранного цикла. | 0 |
| **MB\_CIP\_Edit\_T\_ACID** | UINT | 532 | HMI↔PLC | Оператор | CIPManager | Время подачи кислоты (сек). | 0 |
| **MB\_CIP\_Edit\_T\_PUMP** | UINT | 533 | HMI↔PLC | Оператор | CIPManager | Время циркуляции (сек). | 0 |
| **MB\_CIP\_Edit\_T\_DRAIN** | UINT | 534? (likely) | HMI↔PLC | Оператор | CIPManager | Время слива (сек). (В CSV snippet 533 was PUMP, 534 presumably DRAIN). | 0 |
| **MB\_CIP\_Edit\_Select** | UINT | 535? (or 529?) | HMI→PLC | Оператор | CIPManager | Номер редактируемого цикла CIP (1–7). Оператор выбирает, CIPMgr знает куда сохранить. | 1 |
| **MB\_CIP\_Commit** | BOOL (bit) | (569/570?) | HMI→PLC | Оператор | CIPManager | Команда зафиксировать изменения CIP параметров (вместо LiveDelay). Если реализация автоматически, может не использоваться. [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ]. | 0 |
| **MB\_CommFlags** | UINT | 562 | PLC→HMI | Системно (Comm) | HMI (иконки связи) | Флаги связи: возможно b0=Связь с HMI ок, b1=Связь с ПЧ ок. PLC может выставлять/сбрасывать. Пока не реализовано явно. | – |
| **MB\_HMI\_Heartbeat** | UINT | 543 | HMI→PLC | HMI (timer)【11†】 | FB\_Main (monitor) | “Сторожок” от панели. HMI каждую секунду увеличивает. Main проверяет изменение. | 0 (inc) |
| **MB\_PLC\_Heartbeat** | UINT | 544 | PLC→HMI | FB\_Main【12†】 | HMI (monitor) | “Сторожок” от ПЛК. Main также инкрементирует раз в сек. Панель контролирует изменение (связь). | 0 (inc) |

**3.4 Внутренние (Ram) переменные и флаги:**

(Это глобальные переменные, не привязанные к физическим IO или Modbus, используемые для связи между FB и хранения состояния. Включаем сюда ALM-флаги, режимы Mode\_xxx, параметры PAR\_xxx, и проч.)

**Аварийные флаги (Alarms):** Все они типа BOOL, область Ram, не retain (сбрасываются при перезапуске). Обновляются в коде по событиям, а MB\_Alarms (Modbus) формируется как битовая маска из них. Биты 0–15 маски соответствуют:

* **ALM\_SensorProduct\_Break** (бит0) – Обрыв датчика температуры продукта. Устанавливается Main, если AI\_TempProduct < TEMP\_MIN\_C после фильтрации (подозрение на обрыв цепи). При срабатывании: TempManager отключает нагрев, PasteurMgr может прервать цикл.
* **ALM\_SensorProduct\_Short** (бит1) – Короткое замыкание датчика продукта. Условие: AI\_TempProduct > TEMP\_MAX\_C (выше физически возможного) = КЗ. Реакция аналогично Break.
* **ALM\_SensorJacket\_Break** (бит2) – Обрыв датчика рубашки.
* **ALM\_SensorJacket\_Short** (бит3) – КЗ датчика рубашки.
* **ALM\_MixerFault** (бит4) – Авария мешалки (привода). Условие: DI1=0 **или** ПЧ в состоянии Fault. Устанавливается Main/VFDManager. Реакция: oстанов мешалки (ActuatorMgr BLOCK\_AllOff или отдельная логика).
* **ALM\_HeaterFault** (бит5) – Авария системы нагрева. Условие: DI2=0 (сработал термостат). Reакция: BLOCK\_HeaterOff=TRUE (ActuatorMgr отключает DO6-8), PasteurMgr может прервать цикл, требовать вмешательства.
* **ALM\_PowerFail** (бит6) – Авария: потеря питания в автоцикле. *Неясно, как задается.* Возможно, intended: если PLC перезапустился будучи в середине PasteurActive (STAT\_AutoActive флаг = Yes retained), то устанавливает эту аварию, чтобы оператор знал о срыве процесса. STAT\_AutoActive (retain) у нас есть. Если он TRUE при старте PLC, значит питание потерялось во время авто. Тогда ALM\_PowerFail=TRUE, все режимы сброшены. (Это можно сделать в Main при инициализации). |
* **ALM\_PIDBadParams** (бит7) – Авария: неверные параметры ПИД-регулятора. Устанавливается TempManager, если PAR\_UsePID=TRUE, но регулятор не настроен (например, Kp=0 или разгон неработоспособен). Реакция: TempManager переключается на гистерезис, или просит проверить настройки.
* **ALM\_Reserved8** (бит8) – Резерв.
* **ALM\_Reserved9** (бит9) – Резерв.
* **ALM\_ValveAlk\_Fault** (бит10) – Резерв/авария клапана щелочи. (Например, нет потока, или заклинило, но у нас нет датчика. Пока не используется.)
* **ALM\_ValveAcid\_Fault** (бит11) – Резерв/авария клапана кислоты.
* **ALM\_NoWaterJacket** (бит12) – Авария: нет уровня воды в рубашке. Условие: DI3\_LevelJacket=0. Reакция: отключить нагрев (чтобы не сжечь ТЭНы).
* **ALM\_PasteurAbort** (бит13) – Преждевременный останов пастеризации. Устанавливается PasteurManager, если цикл прерван до завершения (по Stop команде или аварии). Требует внимания оператора (цикл не выполнен полностью).
* **ALM\_CIPAbort** (бит14) – Преждевременный останов CIP. Аналогично, CIP не доведен до конца.
* **ALM\_Reserved15** (бит15) – Резерв.

Кроме маски 16 бит, есть отдельные флаги: - **ALM\_NoWater** (BOOL, вне основной маски) – Датчик потока CIP показал отсутствие воды. Выставляется CIPManager при DI6=0 во время, когда должна быть жидкость. Отображается, возможно, отдельным индикатором или тоже вклюаемым в MB\_Alarms (можно занять бит10 или 11 вместо ValveFault). Пока оставим раздельно. При ALM\_NoWater: CIPAbort.

**Флаги режимов и состояний:** - **MODE\_Current** (INT) – текущий активный режим (0 Idle,1 Pasteur,2 Manual,3 CIP,4 Dispense,5 Recipe). Обновляется Main, можно отправлять на HMI (SystemStatus did bits). - **MODE\_PasteurActive**, **MODE\_ManualActive**, **MODE\_CIPActive**, **MODE\_DispenseActive** (BOOL) – флаги активности режимов (Ram). Устанавливаются Main, используются ActuatorMgr (входы) и HMI (индикаторы). - **STAT\_AutoActive** (BOOL, Retain) – признак, что система находилась в автоматическом цикле (Pasteur/CIP/Dispense) до выключения питания. Используется для ALM\_PowerFail (см. выше). Main при старте: если STAT\_AutoActive=TRUE (с предыдущего запуска), поднять ALM\_PowerFail. - **STAT\_HeatingActive** (BOOL) – Есть включенный нагрев (DO6/7/8). Может устанавливаться ActuatorMgr, если хоть один HeaterCmd true. Используется для индикации (можно лампу “нагрев включен”). - **STAT\_CoolingActive** (BOOL) – Охлаждение активно. Может означать, что клапан охлаждения DO1 открыт. ActuatorMgr или PasteurMgr может ставить TRUE при Pasteur\_CoolingValve\_ON. Для отображения. - **STAT\_MixerRunning** (BOOL) – Мешалка работает (фактически). **В текущей реализации не устанавливается!** Должен быть TRUE, если мешалка действительно вращается. Можно определять: либо если DO3 OR DO4 = TRUE (для прямого управления), либо по VFD\_Status (1 or 2 = running). Зададим: Main: STAT\_MixerRunning = (PAR\_UseVFD? (VFD\_Status=1 or 2) : (DO3 or DO4)). Требуется внедрить. Пока этот флаг всегда 0 → [правка: реализовать в Main]. - **STAT\_HeatTimeout** (BOOL) – Истек лимит времени нагрева. Устанавливается PasteurMgr, если время разгона превысило SP\_HeatTime. Когда TRUE, цикл на паузе (ждет указаний). Сбрасывается при SkipHeat или Stop. (Может HMI показывает “Не достигнута температура”). - … Другие STAT\_x (не явлены больше, пропустим).

**Команды (Ram, от HMI), кроме Mask:** - **CMD\_SkipHeatStage** (BOOL) – Пропуск этапа нагрева. Если TRUE (оператор нажал соответствующую кнопку), PasteurManager сразу переходит к выдержке, игнорируя недогрев. После обработки, Main сбрасывает. - **CMD\_ResetAlarms** (BOOL) – Сброс аварий. Устанавливается, когда оператор нажал сброс. Main при этом очищает ALM\_ *флаги (кроме тех, что физически еще активны). Возможно, SystemStatus.b3 toggled. -* *CMD\_NextDose* *(BOOL) – Запрос следующей порции. Обрабатывается DosingManager, если UsePortionButton=TRUE – имитирует нажатие внешней кнопки. После старта новой порции – сбрасывается. -* *CMD\_Stop* *(BOOL) – “Аварийный стоп” розлива?. Возможно, это глобальная команда стоп (в CSV: CMD\_Stop; “Аварийный стоп”). Если HMI имеет кнопку E-Stop, или пауза. В code, not used because we rely on DI7. Could be intended as "software E-stop" or "Pause" (but they named Resume separate). -* *CMD\_Resume* *(BOOL) – “Продолжить после паузы”. Возможно, пара к CMD\_Stop – если CMD\_Stop делал паузу процесса (вдруг, Pause?), а Resume продолжает (например, после HeatTimeout или manual pause). Not clearly used. -* *CMD\_RestSoftFlow*\* (BOOL) – Сброс счетчика разлива. (Вероятно, опечатка, имелось в виду ResetSoftFlow). Устанавливается, если оператор нажал сброс литража. DosingManager при этом дергает SoftFlowmeter.Reset. Сбрасывается Main.

**Параметры (Ram, retain):** (частично уже перечислены как PAR\_x) - **PAR\_UseMixerReverse** (BOOL, Yes retain) – Разрешить реверс мешалки. Не используется явно; возможно, если FALSE – ActuatorMgr может игнорировать Man4 (REV). - **PAR\_DispenseEnabled** (BOOL, Yes) – Включить режим розлива. Возможно, чтобы можно было отключать блок розлива целиком (для небольших систем). Если 0 – DosingManager должен игнорировать команды. (Not used anywhere in code currently). - **PAR\_DosingEnabled** (BOOL, Yes) – Разрешить порционный розлив. Тонко: DispenseEnabled vs DosingEnabled, быть может: - DispenseEnabled=0 – нельзя входить ни в DispCont ни DispDose, - DosingEnabled=0 – разрешен только непрерывный, но не порционный. - Такие параметры можно учитывать в Main: если Disabled – не давать активировать RunReq. Пока, считаем эти параметры есть, но не задействованы (можно предусмотреть: Main игнорирует Start DispDose if DosingEnabled=0). - **PAR\_PumpFlowRate** (Float, Yes) – Производительность насоса розлива, л/с. Используется SoftFlowmeter. - **PAR\_UsePortionButton** (No retain, config) – (в ParMask, b2) – см. 1.3. - **PAR\_UseGunSensor** (No retain, config) – (ParMask b3). - **PAR\_UseSteam** (Yes) – (Ram param) – см. 2.3. - **PAR\_HeaterGroup2Enabled** (Yes), **PAR\_HeaterGroup3Enabled** (Yes) – см. 2.3. - **PAR\_StoreEnabled** (Yes) – см. 2.4. - **Par\_UsePid** (No) – (в CSV: Par\_UsePid; likely same as PAR\_UsePID from ParMask b0). - **Hyst\_HeatRequest** (No) – см. 2.3, internal flag for hysteresis demand (set in TempMgr when heating). - **CIP\_ModeMask\_Full** (UINT, Yes), CIP\_ModeMask\_Alk, Acid, Rinse – см. 2.5. - **CIP\_C#\_T\_HOT, T\_COLD, T\_ALK, T\_ACID, T\_PUMP, T\_DRAIN** (UDINT each, Yes) – 7 sets (Cycle1..7) – см. 2.5. - **CIP\_C#\_MixerStage1** etc (BOOL, Yes) – флаги включения мешалки на этапе (CSV snippet shows CIP\_C1\_MixerStage1 etc.). CIPMgr uses them to set CIP\_Mixer\_ON on specific substage, e.g., flush stage1 if CIP\_C1\_MixerStage1=TRUE. - **CIP\_DrainOnStopSec** (UDINT, Yes) – время слива при корректном стопе (CIP). - **CIP\_ChemConflictPolicy** (UDINT, Yes) – политика конфликта реагентов. - **CIP\_LiveWriteCommitDelay\_s** (UDINT, Yes) – задержка применения изменений uставок CIP. - ... others perhaps.

*(Примечание: Все retain-переменные должны быть инициализированы разумными дефолтами и записаны в доку. Если каких-то нет – нужно решить, либо добавить как параметры, либо убрать).*

**Single-writer (Единственный писатель) map:** Ниже перечисляем основные переменные и модуль, который единолично их изменяет: - Все DO/AO – только ActuatorManager. - Все ALM\_ *флаги – только FB\_Main (кроме ALM\_NoWater, ALM\_PIDBadParams – TempMgr/CIPMgr). - Mode\_*Active – только FB\_Main. - Mode\_*RunReq – только FB\_Main. - Pasteur\_CircPump\_ON, etc. – только PasteurManager (VarOutput). - CIP\_x\_ON – только CIPManager. - Dosing\_FillPump\_ON, PortionDone – только DosingManager. - Temp\_DOCmd, Temp\_AO\_HeaterPower – только TempManager. - VFD*\*Word, VFD\_Fault – только VFDManager. - SoftFlowmeter.VolumeTotal – только SoftFlowmeter (Reset by Dosing).

В ходе проверки, не выявлено случаев записи одной переменной в нескольких местах, кроме: - *Pasteur\_HeatPowerReq mismatch:* Pasteur outputs BOOL, Actuator expects REAL. Будет исправлено, тогда тоже единственный писатель (Pasteur). - STAT\_MixerRunning – вообще не писалось. Определим: будет писаться Main (1 источник). - ALM\_NoWater – только CIPManager. - CMD\_ResetSoftFlow – HMI->Main sets it, then Main->SoftFlow resets volume. (So Main writes SoftFlow.Reset). - ParMask – bits read by Main, some bits like portionDone (b16) might be written by Dosing? If we follow initial idea: - Possibly portionDone (Ram) could map to ParMask b16 if HMI expects in same register. But we have separate variable, no need to double. - We'll assume no conflict there.

*(Cross-reference CSV in app will detail each variable's write and read usage, marking those without writer/reader. Основные проблемы по XREF были описаны в начале сводки.)*

## 4. Правила и инварианты системы

Здесь суммируем ключевые ограничения и соглашения, которые **обязательно** должны соблюдаться в коде (и проверены):

* **Single-Writer Principle:** Каждая глобальная переменная, которая влияет на выходы или хранит состояние, должна присваиваться (писаться) только в одном месте в программе. В частности:
* **Все DO и AO** – только внутри FB\_ActuatorManager. **Никакой другой блок не должен напрямую делать DOx:=... или AOx:=...**. В текущей реализации это соблюдено (поиск по исходникам показал отсутствие присваиваний DO/AO вне ActuatorManager【40†】). Это инвариант: если потребуется изменить выход (например, отключить нагрев при аварии), следует устанавливать соответствующий флаг (как BLOCK\_HeaterOff) и обрабатывать его внутри ActuatorManager, а не трогать DO напрямую.
* **Команды на нагрев** (включение групп нагревателей) – только FB\_TemperatureManager. Ни PasteurManager, ни Main не должны сами включать DO6-8. PasteurManager лишь выдает разрешение EN\_Heat, а TempMgr на его основе решает, когда включать DO6-8. Проверка: PasteurManager раньше имел влияние (HeatPowerReq), но мы исправляем к Bool EN\_Heat, который TempMgr читает.
* **Выбор направления/скорости мешалки** – при PAR\_UseVFD=TRUE, управляется только VFDManager (через команды CmdWord), при PAR\_UseVFD=FALSE – ActuatorManager (DO3/DO4/AO2). Другие модули (Pasteur/CIP) только выставляют флаги Mixer\_ON, CIP\_Mixer\_ON; ActuatorManager сам решает, как реализовать (DO или VFD).
* **Флаги ALM\_**\* – у каждого аварийного флага один ответственный модуль, обычно FB\_Main (например, датчики), или специфический (TempMgr для PIDBadParams). В случае, если условие тревоги комплексное, лучше централизовать в одном месте. Например, ALM\_MixerFault = TRUE либо от DI1 либо от VFD\_Fault – сделано в Main (комбинируя условия).
* **Параметры (PAR\_)** – по месту тоже не переписываются из разных мест. Обычно только Main читает из Modbus и сохраняет Ram. Например, PAR\_UsePID – только Main (ParMask decode). Другие блоки только читают.
* **Глобальные команды (CMD\_)** – формируются только Main по Modbus битам. Остальные блоки на них реагируют. Искл.: CMD\_NextDose – может приходить как Modbus bit, Main делает Ram bool, DosingMgr сбрасывает его после использования (или Main).

Таким образом, избегается ситуация, когда два блока “тянут” один сигнал в разные стороны. **Если в текущем коде обнаружится, что один сигнал пишется в двух местах – это ошибка, требующая устранения.** (В приложении XREF.csv отмечены такие случаи, пока найден Pasteur\_HeatPowerReq как mismatch и STAT\_MixerRunning (никто не писал).)

* **Приоритеты режимов:** (см. 1.2 Матрица) – Ручной > CIP > Pasteur > Dosing.
* Если активен Manual, все авто режимы должны быть неактивны. ActuatorManager уже отдаёт приоритет ручным сигналам【41†】. Но также следует гарантировать, что Auto FB не выполняют лишних действий: если оператор перевёл в Manual во время пастеризации, PasteurManager всё равно может “думать”, что цикл идет. Это не критично, т.к. выходы не применятся, но нежелательно (таймеры будут тикать). Решение: при входе в ManualActive, Main отзывает RunReq у Pasteur/CIP (прерывая их).
* Одновременно только один авто-режим может получать RunReq. Main реализует через ActiveMode\_UD. Необходимо, чтобы если, например, CIPFull\_RunReq и Pasteur\_RunReq одновременно TRUE (теоретически с панели), Main выберет CIP (см. код)【49†】 и **игнорирует Pasteur\_RunReq до конца CIP**.
* Переключение режимов должно обнулять/сбрасывать состояния предыдущего:
  + После CIPActive=TRUE, если был PasteurActive – его сбрасываем (PasteurActive=FALSE, ActuatorMgr отключит пастеризационные выходы).
  + Если CIP закончился, ActiveMode->0, все CIPOutputs off, CIPMgr resets, дальше можно запускать след. режим.
  + Никакие два режима не должны “бороться” за один механизм одновременно.
  + Как видно, ActuatorMgr с помощью Mode\_xxx\_ACTIVE в условиях IF обеспечивает, что, напр., DO1 используется только Pasteur (Manual) и никак CIP (CIPActive→ DO1=FALSE)【42†】. Подобное для DO2, DO5 и т.д.
  + Искл.: Mixer DO3, DO4 – используется Pasteur и CIP. Но ActuatorMgr при CIPActive включает DO3 по CIP\_Mixer\_ON, а при PasteurActive – по Pasteur\_Mixer\_ON. Если вдруг кто-то оставил Mixer\_ON=TRUE из Pasteur, а CIP уже начат – CIPActive ветка приоритетнее, но PasteurActive в идеале должен стать FALSE. В Main, при CIP start, PasteurRunReq снимается → PasteurActive=FALSE. Так и будет.
  + “Manual wins”: ManualActive=TRUE → Actuator полностью не смотрит на Mode\_Pasteur/CIP (они могут висеть TRUE, но else-if Manual first, so auto ignored). Это сделано【41†】. Также, ManualActive может устанавливаться даже если ActiveMode !=0 (в нашем решении – возможно ввод отдельным битом). Считаем, что Manual selection приводит к обнулению auto-run (да, повторяем, но важно).
* **Безопасное состояние при потере связи:**
* **HMI связь потеряна:** Panel пропала – система продолжает текущий цикл, но оператор не может управлять. В зависимости от требований, можно:
  + Либо автоматически остановить автоцикл (продукт может пережариться – но у нас TempMgr автономно справится, CIP может тоже по таймерам кончить).
  + Не стоит автоматически прекращать – лучше дать завершить сработав логику. Но (!) если происходит авария, оператор не сможет отправить Stop.
  + У нас E-Stop аппаратный. Так что, если HMI пропал, пусть ПЛК доработает цикл до конца (AutoActive).
  + Тем не менее, мы вводим:
  + Если HMI offline, блокируем **запуск новых режимов**. (ActiveMode не переключать).
  + STAT\_HMI\_LinkLoss (bit in MB\_CommFlags) – можно отобразить.
  + В документе: *При потере связи с панелью операторской, ПЛК завершает текущий цикл, но новые команды не принимаются; отображается сигнализация потери связи.*
* **VFD связь потеряна:**
  + Если PAR\_UseVFD=TRUE и VFD offline – мешалка управления не получает.
  + ПЛК обнаружив, должен:
  + Установить STAT\_VFD\_Online=FALSE (Ram).
  + Выставить ALM\_MixerFault=TRUE (рассматриваем потерю связи как авария привода) – хотя отдельный признак можно сделать, но чтобы не добавлять новые ALM, используем существующий.
  + ActuatorMgr – уже косвенно мешалку отключил, т.к. ModeManualActive or auto wouldn't set commands if offline (Main could handle).
  + Main/Actuator:
    - Manual tries to use VFD – if offline, drive not respond. ПЛК может блокировать ManualMan3/Man4 (e.g., Actuator check STAT\_VFD\_Online in manual branch – if false, DO3/4 off).
    - CIP/Pasteur auto tries – similarly if offline, CIP\_Mixer\_ON commands won't have effect, motor stays off physically.
  + Good to ensure safety: If VFD offline and CIP requires mixing, skip mixing or abort CIP? CIPMixer presumably optional, so CIP can proceed without mixing (less effective but okay).
  + Pasteur mixing off – risk product uniformity – operator should notice ALM\_MixerFault.
  + So:
    - Temp measure: If VFD comm lost mid-run, raise alarm, but let process continue, albeit without mixing.
    - If mixing absolutely needed (milk pasteurization), operator may abort and fix. Up to them.
  + В частотнике AD30, возможно, есть встроенный параметр “Stop on comm loss” – но мы не рассчитываем на это, лучше ПЛК не посылает ложных команд.
  + Implementation:
  + Main: if no feedback from VFD for >2 sec, set STAT\_VFD\_Online=FALSE, ALM\_MixerFault=TRUE.
  + ActuatorMgr (ManualActive):
    - Add: if Mode\_Manual\_ACTIVE and PAR\_UseVFD and not STAT\_VFD\_Online -> игнорировать Man3/Man4 (оставить DO3/4 off).
  + Already, if offline, VFD\_CmdWord we might stop sending run commands (or we still send and hope if regained, drive picks up? But regained improbable spontaneously).
  + Fine to continue sending, but ALM informs operator.
* **Датчики температуры:**
  + Если обрыв – TempMgr перестает греть (prodC\_ok fixed, ALM on), PasteurMgr abort. Already done partly: ALM triggers.
* **Насос розлива/SoftFlow:**
  + Потеря связи не влияет – SoftFlow internal.
* **Прочие устройства:**
  + DI7 E-Stop – аппаратный, covered.
  + DI6 flow – if no flow CIP abort as above.
* **Фильтрация, гистерезис, ΔT защита:**
* TempManager:
  + Фильтр expo (α~0.2).
  + Гистерезис нагрева Δ ~2°C (SP\_SmoothDiff).
  + ΔT защита: не греть, если рубашка > продукт + 15°C (во избежание пригорания).
  + Min On/Off ~10 с (для контакторов) – [внедрить].
  + CIPManager:
  + Гистерезис охлаждения: SP\_SmoothDiff применен【79†】.
  + ΔT CIP: не используется (CIP medium can be hot separate).
* SoftFlow:
  + Потенциальная погрешность ±1 сек \* Rate Л/с (e.g. 0.5 L if 0.5 l/s).
  + If needed more accurate, refine code – [optional].
* Mixer:
  + Непрерывная работа – control by dryness as above.
* **Нейминг и разделение ответственности:**
* Приняты префиксы: PV\_ – Process Value (текущие измерения), SP\_ – SetPoint, PAR\_ – параметр, CMD\_ – команда, Mode\_ – флаг или запрос режима, ALM\_ – аварии, STAT\_ – статусы, DI/DO/AI/AO – физические IO, MB\_ – Modbus.
* Prefix by subsystem: Pasteur\_\* – сигналы, идущие от PasteurManager (к Actuator), CIP\_\*, Dosing\_\*, Temp\_ (TemperatureManager), VFD\_\*.
* Suffixes:
  + \_ON – команды включения механизмов (обычно BOOL, идет в Actuator DO),
  + \_Cmd – команды как выход TempMgr (BOOL),
  + \_RunReq – запрос запуска режима (BOOL),
  + \_ACTIVE – флаг активного режима (BOOL),
  + \_x10 – величина в десятых (INT),
  + \_Pct – процент, \_Hz – частота, etc.
  + \_Mask – битовая маска (UINT).
  + \_Raw – необработанное значение (REAL or BOOL?), здесь использовано в PV\_TempProduct\_Raw (REAL).
  + We see confusion in CSV where Raw was Boolean (incorrect) – to fix: rename PV\_TempProduct\_Raw (Real) vs maybe they intended a bool "fresh reading"? Not necessary. We'll align type with usage (Real).
* **Запрет прямого управления выходами вне Actuator** – уже оговорено.
* **"Manual wins"** – реализовано.
* **Single-writer enforcement** – см. выше. (In code review, ensure no duplicates).
* **Разделение FB responsibility:**
  + PasteurMgr – только технология пастеризации, он не дергает выходы напрямую.
  + CIPMgr – только CIP, etc.
  + Главный bridging, actuators unify actual hardware.
* **Temporary alias bridging and renaming:**
  + We noticed e.g. Pasteur\_HeatPowerReq conflict.
  + **Plan:** Rename PasteurManager.HeatingPowerReq (BOOL) to Pasteur\_HeatEnable for clarity. Also adjust Actuator input to BOOL Pasteur\_HeatEnable (instead of REAL). This solves mismatch. Or keep name but fix type to BOOL and update Actuator expecting BOOL.
  + Standardize PV\_TempProduct\_Raw type to REAL (fix CSV).
  + SP\_DoseVolume to REAL (fix CSV).
  + Possibly unify ALM\_NoWater signals (make it part of ALM\_Mask bits or separate clearly).
  + Provide alias if needed:
  + During code update, define at top of Actuator or Main: e.g., Pasteur\_HeatEnable AT PasteurManager.HeatingPowerReq; if linking by address needed (depending on Owen environment).
  + But since it's same code project, easier: change definitions and recompile.
* **Zoo of names:** Current naming mostly consistent as above, small corrections:
  + DI6 named D6I\_ProductFlow in CSV (typo "D6I"), should be DI6\_ProductFlow.
  + Par\_UsePid vs PAR\_UsePID – unify case (prefer PAR\_UsePID).
  + MB\_ModeCurrent\_in (539) and MODE\_Request (540) were reserves – can remove or leave not used.
  + Some signals appear duplicates (PAR\_DispenseEnabled vs Mode\_DispCont?), but likely not used.
  + We list those as not used [ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ], possibly remove in final code to avoid confusion.

## 5. План тестирования и критерии приёмки

Для приемки обновленного ПО необходимо провести всесторонние испытания, имитируя как программно, так и на реальном оборудовании, все ключевые сценарии. Разделим тестирование на два этапа:

**5.1 Тесты на модели (без оборудования):** Цель – проверить логику программы в отладчике OwenLogic, с имитацией сигналов (чрез принудительную установку DI/Modbus переменных), наблюдая переменные и переходы состояний.

* **Инициализация и Idle:** После загрузки ПО и холодного старта все выходы должны быть в 0, ALM\_ флаги = FALSE (кроме тех, что связаны с сохраняющими авариями). Проверить: DO/AO = 0 (в VarMonitor), MB\_Alarms=0, Mode flags = FALSE. Если STAT\_AutoActive retain был TRUE, то ALM\_PowerFail = TRUE (bit6 mask) – проверить это, затем сбросить (ResetAlarms).
* **Ручной режим управление:** Установить ManualActive (в Main, например, смоделировать MB\_SystemStatus.b0=1 если предусмотрено или просто установить Mode\_ManualActive=TRUE).
* Вручную выставлять Man1..Man14 = TRUE по очереди и проверять, что соответствующие DO срабатывают немедленно (и только они).
* Например, Man6\_Heater1=TRUE => DO6=TRUE, DO7/8 остаются FALSE (если другие ManX не нажаты).
* Нажать Man3 (мешалка вперёд) – если PAR\_UseVFD=FALSE, убедиться DO3=TRUE, AO2=10V (100%). Если PAR\_UseVFD=TRUE, DO3 остается FALSE, но VFD\_CmdWord = 1 формируется.
* Комбинировать: Man3 и Man4 вместе -> ожидаем DO3=TRUE, DO4=TRUE (если мешалка способна?), но лучше аппаратно это невозможно; программа установит оба (нет явной блокировки) – отметить, что это возможно, но физически либо оба контактора не могут включиться, либо ПЧ проигнорирует двойную команду. В будущей версии – можно добавить приоритет (например, если оба, брать FWD).
* Отжать все Man – убедиться выходы вернулись в 0.
* Проверить блокировки: Смоделировать DI7\_EStop=0 (находясь даже в ручном): BLOCK\_AllOff должно стать TRUE, все DO=FALSE немедленно (несмотря на Man).
* Смоделировать DI2\_HeaterFault=0: BLOCK\_HeaterOff=TRUE => даже если Man6/7/8 нажаты, DO6-8 должны сброситься в FALSE (проверить).
* DI3\_LevelJacket=0 аналогично.
* **Пастеризация цикл (номинальный сценарий):**
* Задать необходимые параметры: SP\_HeatTemp (например, 750 = 75.0°C), SP\_HoldTime (e.g. 30 s), SP\_CoolTemp (e.g. 300 = 30.0°C). ParMask: UsePID=TRUE/False (протестовать оба случая).
* Установить Mode\_Pasteur\_RunReq=TRUE (имитируя нажатие “Старт пастеризации”).
* Убедиться: PasteurManager перешел в Stage=Heat, PasteurActive=TRUE. Actuator: DO2=TRUE (насос), DO3=TRUE (мешалка), DO1=FALSE (охлаждение выкл). EN\_Heat=TRUE (TempMgr видит).
* Смоделировать рост температуры: В отладчике вручную повышать PV\_TempProduct\_Raw каждую секунду. Например, от 20°C до 80°C постепенно. Следить:
  + TempMgr (UsePID=FALSE): Он должен включать DO6-8 = TRUE при EN\_Heat. По достижении ~75°C, около SP, TempMgr должен начать отключать нагрев (Hyst logic): когда PV >= 75°C + Δ (допустим Δ=2°) = 77°C, Hyst\_HeatRequest→FALSE => DO6-8 = FALSE. (Если PID=TRUE: DO6-8 PWM могут дергаться, сложно увидеть на пошаговой, но AO\_HeaterPower будет <100, соответственно DOx).
  + PasteurMgr: как только PV\_TempProduct >= SP\_HeatTemp (75°C), он должен переходить в Stage=Hold. (В PID случае, PV может плавно подойти; в гистерезисе – может немного перегнать). Проверить: Mode\_PasteurActive остается TRUE, но возможно он выставит отдельный индикатор, хотя Stage updated. In absence of direct indicator, check internal stage variable if possible.
  + На переходе Hold: PasteurManager запускает таймер hold =30с (в реальности fast-forward).
* Дождаться завершения выдержки (можно руками установить t\_hold=0). PasteurMgr -> Stage=Cool.
  + EN\_Heat должно стать FALSE (TempMgr turn off DO6-8).
  + Actuator: DO1\_CoolingValve=TRUE, DO2 circ pump likely still TRUE, DO3 mixer TRUE.
  + PV\_TempProduct начнет падать (смоделировать: плавно уменьшать Raw).
  + Проверить логику гистерезиса охлаждения: пока PV > 30°C, ValveCold=TRUE; как только PV\_рубашки <= 28°C (SP\_CoolTemp -2°), PasteurMgr временно закроет DO1 (CoolingValve\_OFF)【79†】, дайте PV\_product догреться рубашку, потом опять откроется – имитировать, убедиться.
  + Когда PV\_product <= 30°C (SP\_CoolTemp), PasteurMgr завершает охлаждение: Stage = Done.
  + Ожидаем: DO1=FALSE (закрыл), DO2, DO3=?? – можно либо оставить до конца или Off. Логично: при Done, PasteurMgr снимает PasteurActive (RunReq=FALSE), поэтому ActuatorMgr Else branch: DO2, DO3 off.
  + Убедиться: Mode\_Pasteur\_RunReq стало FALSE (Main поставит), Mode\_PasteurActive=FALSE. Actuator: DO2,3 off.
* Итого, проверить, что цикл прошел: MB\_SystemStatus (bit for auto) – was 1, now 0; MB\_Alarms – нет аварий (если все нормально).
* В процессе, проверить инварианты:
  + Temp overshoot/undershoot: если PID mode – AO\_HeaterPower менялся как ожидалось (например, сначала 100%, под конец снизился).
  + HeatTimeout scenario: Сделать SP\_HeatTime\_min коротким (1 мин), симулировать медленный рост или отсутствие роста: AI\_TempProduct остается низкой >1 мин -> STAT\_HeatTimeout=TRUE, PasteurMgr должен не перейти в Hold сам. Можно потом подать CMD\_SkipHeatStage, проверить, что он перешел к Hold (несмотря на недостижение SP).
  + SkipHeat: на этапе Heat, когда EN\_Heat=TRUE, нажать CMD\_SkipHeatStage – PasteurMgr должен немедля: EN\_Heat->FALSE (отключит нагрев), перейти к HoldTimer=0? или сразу Stage=Cool – по реализации у нас: skip heat -> Stage=Hold (0 s hold?), но можно трактовать как сразу Cooling. Вероятно, skip heat значит “не ждать достижения T, сразу начинай выдержку (если можешь)”. Если продукт недогрет – ну, hold при низкой T, получится недопастеризация, но это на совести оператора.
  + Проверить: skip triggered, TempMgr EN\_Heat все равно false – т.е. больше не греет, holdTimer пошел.
  + Через SP\_HoldTime цикл перейдет к Cool.
  + Stop Pasteur (Abort):
  + Смоделировать на середине Hold: MB\_CmdMask1.b1 StopPasteur =1. Main => Mode\_Pasteur\_RunReq=FALSE (?), PasteurMgr -> Detect Stop, ALM\_PasteurAbort=TRUE.
  + Actuator: ManualActive? no, just auto off because RunReq off.
  + Check: MB\_Alarms bit13 set. PasteurActive=FALSE, DOs off, Cool not done maybe, but abort stops process (product not pasteurized).
  + Check CIP if any effect – none since CIP not running.
* **CIP режим (Full) тест:**
* Инициализировать: задать CIP\_C# times небольшие, ModeMaskFull=127 (по умолч.), CIP\_Full\_RunReq=TRUE.
* CIPManager:
  + CIPActive=TRUE, Actuator: DO13 PumpCIP=TRUE, CIPStagesMask=127 out to HMI, Step\_CIP=1.
  + В Cycle1: если задали T\_COLD=5s, T\_HOT=5s:
  + DO9 Cold open 5s, затем close; DO10 Hot open 5s, close.
  + DO13 Pump likely ON from start through flush (вода протекает).
  + DI6\_ProductFlow: set to 1 to simulate flow. If leave at 0: after 1-2 sec CIPMgr should ALM\_NoWater=TRUE, abort CIP:
    - Test that: DI6=0 at flush start -> CIPMgr triggers ALM\_NoWater, CIPAbort.
    - Actuator: DO9 closed, DO13 off, DO14 Drain opened 30s (CIP\_DrainOnStop=30 – simulate).
    - MB\_Alarms bit14 CIPAbort=TRUE, bit others perhaps.
    - Reset alarms, try again with DI6=1.
  + After flush cycle1, CIPMgr goes to T\_PUMP (if set e.g. 3s):
    - CIP\_Pump remains ON, CIP valves all closed, CIP\_Mixer maybe ON if CIP\_C1\_MixerStage1=TRUE.
    - After 3s, then T\_DRAIN cycle1:
    - CIP\_Pump possibly OFF, CIP\_DrainValve DO14 = TRUE for set sec (drain).
    - Confirm: Actuator DO14 on, DO13 off or on? Possibly off to let drain out – check if CIPMgr turns pump off: we would, to drain effectively.
    - After drain, CIPMgr goes next cycle.
  + Cycle2 (Alk): if Mask says included:
  + CIPStagesMask bits show, Step\_CIP now 2.
  + T\_ALK, T\_PUMP, T\_DRAIN, etc. CIP\_Valve\_Alk DO11 cycles open accordingly.
  + CIP\_Pump toggling logic same: Off during drains, On other times.
  + CIPMixer if flagged for stage (maybe CIP\_C2\_MixerStage1 to stir during chemical flush).
  + After all cycles done:
  + CIPActive=FALSE, CIP\_OutStageMask perhaps cleared or left, Step\_CIP=0 or special code.
  + DO9-14 all OFF.
  + ALM – none if all good.
* Test CIP Stop:
  + Start CIP Full, then halfway (say after cycle3) toggle MB\_CmdMask2.b10 CIP\_Stop=1.
  + CIPMgr should abort current stage:
  + Immediately stop pump (DO13=OFF),
  + Possibly open drain (DO14=ON) for CIP\_DrainOnStopSec (30s) to dump contents.
  + After that, CIPActive->FALSE.
  + ALM\_CIPAbort=TRUE (bit14).
  + Confirm outputs: DO9-12 should be closed too.
* Test CIP variants:
  + CIP Alk mode: CIP\_Alk\_RunReq=TRUE (others false) -> CIPStagesMask=0x67, cycles skip acid.
  + Ensure CIPMgr respects mask: cycles 1,2,3,6,7 run, 4,5 skip.
  + Check e.g., no acid valve activation in cycle4 because mask bit3 false -> CIPMgr should jump from cycle3 to cycle6.
  + CIP Rinse mode: CIP\_Rinse\_RunReq=TRUE -> cycles 6,7 only. CIPMgr likely will jump directly to cycle6 as first, then 7, done quickly.
  + CIP multiple triggers conflict:
  + If Full and Alk bits set simultaneously, Main logic ensures just Full (since checks CIP\_Full first).
  + But test: set CIP\_Alk\_RunReq=TRUE, then CIP\_Acid\_RunReq=TRUE while CIPAlk still running? ActiveMode ignoring second (since CIPActive true, won't heed new req).
  + The conflict policy might not come into effect via code, because ActiveMode gating prevents it. So it's fine.
* **Розлив (Dispense) тесты:**
* Порционный режим:
  + Set SP\_DoseVolume (e.g. 5.0 L). PAR\_PumpFlowRate (say 1.0 L/s).
  + Test UseGunSensor on/off, UsePortionButton on/off combinations:
  + Case 1: UseGunSensor=FALSE, UsePortionButton=FALSE (fully auto, no gun check).
    - Set Mode\_DispDose\_RunReq=TRUE -> DosingMgr should immediately start portion.
    - FillPump\_ON=TRUE should go high (Actuator DO5).
    - SoftFlowmeter resets volume to 0 and starts counting.
    - Simulate SoftFlow: either by letting Seconds\_UD increment and Rate=1 L/s, or manually increase VolumeTotal in debug:
    - After ~5 sec (since 5L at 1 L/s), VolumeTotal≥5.0 -> DosingMgr should stop pump: FillPump\_ON=FALSE, PortionDone=TRUE for a moment.
    - Because UsePortionButton=FALSE, DosingMgr should automatically commence next portion:
      * i.e., likely on next cycle, FillPump\_ON toggles back to TRUE (since still EN\_Portion and canStart auto).
      * Check if it does: likely yes if we see pump toggling after fraction of second. If portionDone resets quickly, might blink not visible externally.
      * We can put breakpoints: after portion done, see if next portion started (Volume reset to 0 and FillPump\_ON back to 1).
      * Let it do 2-3 portions to confirm continuous dispensing cycles.
      * Then set Mode\_DispDose\_RunReq=FALSE (stop mode) during an ongoing portion:
      * DosingMgr should immediately stop pump (FillPump\_OFF) mid-portion, not set PortionDone (incomplete).
      * Then portion mode off.
      * Good.
  + Case 2: UseGunSensor=TRUE, UsePortionButton=TRUE (most restrictive):
    - Place DI5\_GunSensor=0 (pistol in place).
    - Mode\_DispDose\_RunReq=TRUE: DosingMgr now awaits external trigger (PortionButton) since UsePortionButton=TRUE, and GunSensor must be true (which it is).
    - Press DI4\_PortionBtn=1 (simulate external button press) -> on that rising edge, DosingMgr should start pump:
    - FillPump\_ON=TRUE, Volume reset, counting.
    - Remove DI4 (0) after pressing (momentary).
    - Now while pouring, simulate mid-run events:
    - Remove DI5 (set GunSensor=1 meaning gun removed) mid-portion:
      * Expect: pump stops immediately (FillPump\_OFF), portion incomplete (Volume maybe 3L of 5).
      * DosingMgr sets pouring=FALSE due to sensor fail, sets awaitTrigger=TRUE to require re-trigger maybe.
      * PortionDone should NOT go true (because not reached volume).
      * Now if Gun back (DI5=0 again):
      * Press PortionBtn again:
        + DosingMgr should probably start a fresh portion from zero or continue the incomplete one? More likely it resets and starts new since original portion considered aborted.
        + It should have maybe reset Volume when aborted or on next start:
        + Need to confirm our design: we would reset integrator at new portion start anyway, so any leftover 3L are lost to record (makes sense, spilled volume).
        + Pump starts again and will aim to do full 5L presumably. If operator wants only remaining 2L to reach total 5 originally, system doesn't track that – they'd have to guess or measure external. That is a limitation but acceptable under emergency stop.
      * Once portion fully delivered second attempt (5L), pump stops, PortionDone=TRUE.
    - If no mid-run issues: after pump stops by volume:
    - FillPump\_OFF, PortionDone=TRUE.
    - Now DosingMgr enters waiting state for next portion (because UsePortionButton=TRUE).
    - Check that pump remains off until user action.
    - Press portion button again (simulate new container ready):
      * Volume should have been reset (should at portion start logic).
      * Pump on again for next 5L.
    - Now simulate stop command or mode off while idle waiting:
      * If Mode\_DispDose\_RunReq=FALSE when not currently pumping (just finished portion), then it should just go idle (no pump to stop, portion done maybe still true a cycle).
      * Or if pressed stop while pumping:
      * This similar to earlier: stops pump mid portion, incomplete, portion mode off, portiondone not flagged.
    - Test portion done output usage:
    - Ideally, portionDone True visible at HMI as lamp or triggers external filler:
      * We verify portionDone goes True in variable watch when portion ended.
      * See that it resets either on next portion or after some seconds:
      * If we have access to code, we might see portionDone being set then cleared.
      * Possibly, in absence of explicit timer, they might rely on portionDone being a short pulse:
        + Could be using rising edge function or so, but probably not used in ST, likely they do manual toggle:
        + e.g., set TRUE when volume reached, and maybe in next scan set FALSE (like a single-scan pulse).
        + That might be too short for HMI; maybe they leave it true and expect operator to reset or it resets at next start.
        + Safer: hold until next portion triggers or mode turned off resets it.
      * We'll note if portionDone remains True after finishing portion (the code snippet didn't show how long).
      * If portionDone remains true and waiting for next, maybe they intended panel or main to clear it.
      * Possibly they map it to ParMask bit16, and Main clears after a second or so.
    - To test, we can just check portionDone variable timeline:
      * If it stays True, that might be an oversight (the user might see portiondone light and maybe resets by pressing NextDose or by starting next portion which would implicitly clear because Reset calls SoftFlow might also clear portiondone? Not directly).
      * If it goes False next scan, our test might miss it unless breakpoints used.
      * Could artificially slow down or instrument code with a small delay just to see, but it's fine if we trust logic.
  + Case 3: UseGunSensor=TRUE, UsePortionButton=FALSE (semi-auto but gun interlock):
    - Mode\_DispDose=TRUE: since no portion button needed, it should start immediately if gun in place.
    - If gun not in place (DI5=1) at start, it should wait (not start pump until DI5 goes 0).
    - Place gun (DI5=0) -> pump starts.
    - Then it will auto restart next portion without waiting for button (since portionButton disabled), as long as gun stays in place.
    - If gun removed between auto portions:
    - When portion ended, portionDone triggers, it tries to start next portion immediately because no button needed, but gun is now absent => condition fails in canStart (GunSensor false).
    - So it will wait until gun placed back to continue.
    - That yields a dynamic pause until sensor regained, then next portion flows.
    - Test removing gun mid-run as before:
    - Should abort current portion.
    - If no portion button, after abort, does it auto restart portion once sensor back? Possibly yes, because from program perspective, EN\_Portion still true, portion not active currently, portionButton not needed, so canStart will become true as soon as GunSensor true again (milk portion incomplete but system doesn't know it was incomplete, it just knows portion was aborted with no portionDone flagged, but it might treat it like portion done? Actually, if aborted due sensor, portiondone remained false so DosingMgr might still consider that portion incomplete – logic in code not clearly defined; likely they treat it as portion aborted and not retried automatically).
    - Possibly DosingMgr might attempt to resume portion spontaneously when sensor returns because still in state "we were pouring but paused"? If code not handling that state, could be glitch:
      * Perhaps they simply stopped pump and set pouring=FALSE, but still in mode and not having portionDone or awaitTrigger, with portionButton off needed anyway no, in no portion button case:
      * If sensor returns, their canStart condition (UsePortionButton false, GunSensor true, portionStop maybe, it might think "we can start a portion because portionActive false and no triggers needed beyond we still in mode"). That would cause it to start a *new portion from scratch* – effectively discarding partially filled previous container and starting a fresh portion measure.
      * That is likely – so aborted portion volume lost, new portion given. Not ideal physically, but software wise plausible.
    - We'll accept that limitation; mention operator should handle any partial fill manually.
* Непрерывный режим:
  + PAR\_UseVFD (irrelevant to dosing).
  + Test Gun interlock:
  + UseGunSensor=TRUE, Mode\_DispCont\_RunReq=TRUE:
    - If gun not present (DI5=1) at start, DosingMgr should not start pump. Check FillPump remains off while DI5=1 even though mode on.
    - Set DI5=0 (gun placed), pump should start.
    - Remove gun (DI5=1) mid-flow, pump stops.
    - Replace gun, since still runreq true, pump likely resumes (should it?
    - In portion we decided not auto resume, but for continuous maybe auto resume because no user action needed to keep pumping).
    - Code currently: canStart for continuous likely not needed, they probably just do FillPump\_ON = EN\_Continuous AND (not UseGunSensor OR GunSensor).
    - If implemented like that, then yes: when gun removed, GunSensor false, expression false -> pump off; when gun returns true -> expression true -> pump on again (so auto resume).
    - That could cause a spurt when reattached – presumably acceptable.
    - Turn off Mode\_DispCont\_RunReq, pump stops.
  + UseGunSensor=FALSE, just continuous mode:
  + Mode on -> pump on continuously, stop -> off. Straightforward.
  + Check SoftFlowmeter usage in continuous:
  + If want to track total pumped:
    - Press CMD\_ResetSoftFlow at start, SoftFlowVolume resets.
    - Let run 10s at 1 L/s, volume ~10 L.
    - Check VolumeTotal increments accordingly (should be exactly time \* rate, given second tick).
    - Press Reset flow mid-run: SoftFlow resets volume to 0 (should see it drop).
    - In continuous, portionDone not used, so fine.

**5.2 Тесты с оборудованием:** После отладки логики, провести испытания на реальном пастеризаторе (или симуляторе с физическими модулями), включая: - **Проверка входов/выходов:** Нажимать реальные кнопки (ручного управления), контролировать срабатывание соответствующих механизмов (клапанов, насосов). Проверить индикацию на HMI (лампы Manual/Auto, состояния). - **Цикл пастеризации с водой:** Налить в емкость воду, задать уставку ~ 60°C, время выдержки 1 мин. Запустить автомат. Замерить: - Включение нагревателей – наблюдать ток/индикаторы. - Достижение ~60°C на датчике – сравнить с показаниями HMI PV\_TempProduct. Убедиться, что выдержка началась (HMI показывает этап). - Время выдержки – по окончании проверить, что HMI переходит к охлаждению. - Подача охлаждающей воды – ощутимо ли охлаждает, продукт температура падает. - Окончание – убедиться, что всё выключилось, сигнал “цикл завершен” (если предусмотрен). - **Имитировать аварии:** - Отключить датчик (вынуть клемму AI) во время нагрева – должна всплыть ALM\_Sensor\_x\_Break, нагрев отключиться, ALM\_PasteurAbort, цикл стоп. - Пустить снова, но отключить насос рубашки (имитировать перегорел – DI1 or just close valve so no heat transfer) – нагрев будет очень медленный; скорее сработает HeatTimeout. - Нажать аварийный стоп (E-Stop) – убедиться, что питание на ТЭНы, насосы снято, цикл прерван. - **CIP мокрый прогон:** - Наполнить систему водой/моющими жидкостями (в емкости + подключить CIP). - Запустить CIP Full: следить, что открываются нужные клапаны, насос гонит (видно по шуму/напору), жидкость переходит между емкостью и канализацией. - Проверить, что по завершении система не оставляет жидкость внутри (финальный слив). - Испытать Stop CIP на середине: убедиться, что сразу переходит на слив и останавливается. - Проверить датчик потока: закрыть подачу воды перед запуском CIP, запустить – система должна через несколько секунд остановиться с аварией “нет воды”. - **Розлив тест (с водой):** - Подключить шланг/“пистолет” к насосу розлива, измерительную емкость. - Порционный режим: задаем объем, запускаем. Пошедший объем сверять с измерительным цилиндром. Скорректировать PAR\_PumpFlowRate при необходимости (например, если перельет систематически на 10% – уменьшить Rate). - Проверить разные варианты: - Нажать/отпустить “пистолет” (имитир. GunSensor) – убедиться, что без “пистолета” насос не льет. - Нажать кнопку порции – одна порция. - Автоповтор (если отключить кнопку) – наливает порции подряд, оператор лишь меняет тару. Оценить стабильность – порции +- допустимая погрешность (в идеале <5%). - Прервать режим в середине порции – увидим, что насос остановился, объем недолит – ок. - Непрерывный режим: - Открыть “пистолет” (GunSensor), нажать Старт непрерывного – насос качает постоянно, пока не остановят. Проверить скорость потока примерно соответствует ожиданию (Rate). - Закрыть “пистолет” (GunSensor=1) – насос остановится, давление упадет (звук). Открыть снова – возобновит. (Предупредить: чтобы не рвануло шланги). - Проверить команду ResetSoftFlow: в режиме непрерывного, пусть нальет ~15 л, нажать ResetFlow – на HMI счетчик обнулится. - **VFD управление мешалкой:** - Если ПЧ подключен и PAR\_UseVFD=TRUE: - Запустить мешалку через HMI: задать скорость %, нажать Mixer Start (Manual or via CIP if CIP requires). - Проверить: DO3 не включается (пускатель не щелкает), а на ПЧ приходят команды: бегущий индикатор “RUN”, заданная частота = (P0-14\*%). - Изменить скорость с панели на лету – частота ПЧ меняется (мотор ускоряется/замедляется). - Реверс: нажать Mixer Rev (Manual) – ПЧ останавливается и стартует в обратную сторону (проверить ток, или флаг на ПЧ). - Авария ПЧ: создать условие (например, перегрузка, или вручную Fault bit). Если нельзя, имитировать отключением RS485: связь потеряна -> через 2 сек ALM\_MixerFault. - Убедиться, что при этом ActuatorMgr не выдаёт DO3, а HMI alarm. - ПЧ Reset: после фиксации Fault на ПЧ (напрямую принудительно), нажать ResetAlarms – VFDManager должен послать CmdWord=7 (Reset) один раз, ПЧ выйдет из аварии. - Продолжить управление.

* **Retain memory тест:**
* Включить какой-то режим (Pasteur mid-cycle), выключить питание ПЛК (симуляция). Включить – убедиться, что ALM\_PowerFail=TRUE.
* Параметры retain (в EEPROM) – изменить, перезагрузить ПЛК – убедиться, что они сохранились (например, PAR\_HeaterGroup3Enabled=0, reboot, остался 0).
* STAT\_AutoActive: перед отключением, Main должен был сохранить True (if was auto). Check after restart that that triggered alarm as above and then it resets to false for subsequent runs (Main should set AutoActive=FALSE after raising alarm to avoid re-trigger if not cleared).

**5.3 Чек-лист сценариев:**

* **Нормальная пастеризация:** Без аварий, с достижением температуры. ✔
* **Пастеризация: недостижение температуры (HeatTimeout):** Прерывание с таймаутом. Оператор решает Skip или Stop. ✔
* **Пастеризация: авария датчика:** Прекращение нагрева, ALM, Abort. ✔
* **Пастеризация: авария нагревателя (термостат):** Нагрев выкл, ALM, Abort. ✔
* **CIP полный цикл:** Все этапы выполнены, без вмешательства. ✔
* **CIP: нет воды:** ALM\_NoWater, цикл Abort. ✔
* **CIP: ручной стоп на этапе щелочи:** Переход в слив, ALM\_CIPAbort. ✔
* **Ручное управление всеми механизмами:** Каждый выход реагирует корректно, не конфликтует. ✔
* **Переход Manual↔Auto:** Взять цикл Pasteur, во время нагрева переключить в Manual – выходы управляются вручную, PasteurActive прерывается. Вернуть Auto – (т.к. RunReq сброшен, он не продолжит) – этот сценарий зафиксировать как некорректный (operator should not do). Но хотя бы, никаких сбоев (ALM\_PasteurAbort). ✔
* **Розлив порционный с параметрами:** Проверить объемы с разной скоростью и объемом (например, 1л, 5л). ✔
* **Розлив: снятие пистолета:** Насос останавливается мгновенно, возобновляется при установке. ✔
* **VFD: изменение скорости, авария связи:** ✔

Все тесты должны быть пройдены. Особое внимание – аварийным ситуациям, чтобы система всегда переходила в безопасное состояние (нагрев и двигатели выключены, насосы остановлены, клапаны закрыты, сливы открыты при необходимости).

В ходе тестирования также собирать открытые вопросы (например, подтверждения по нерешенным моментам, как ALM\_NoWater отобразить или как долго держать PortionDone). Эти вопросы задокументированы в приложении “список открытых вопросов” – их нужно решить с технологом/заказчиком перед финальной прошивкой.

## 6. Перечень доработок и патчей кода

*(В этом разделе перечислим конкретные изменения, которые нужно внести в исходники ST, чтобы привести программу в соответствие с требованиями. Каждый патч – отдельным пунктом, с указанием файла, места вставки/замены, и нового кода.)*

**6.1 Изменения в FB\_Main\_PR102:** - **Инициализация STAT\_AutoActive / PowerFail:** В секции инициализации (перед основным циклом, либо в первом скане):

IF NOT Enable THEN  
 // First scan initializations  
 IF STAT\_AutoActive THEN  
 ALM\_PowerFail := TRUE;  
 END\_IF;  
 STAT\_AutoActive := FALSE;  
END\_IF;

(Это нужно, чтобы при старте, если retain-флаг автоцикла был TRUE, выдать аварию потери питания и сразу сбросить его, чтобы не повторялось.) - **Обработка MB\_CommFlags (связь):** Если использовать рег.562, в Main можно каждые ~5с выставлять бит:

MB\_CommFlags.0 := BOOL\_TO\_UINT(NOT WatchdogHMI) // hypothetical usage

[ТРЕБУЕТ ПОДТВЕРЖДЕНИЯ] – Если не используется, убрать MB\_CommFlags. - **Сброс команд Modbus после обработки:** После формирования reqXXX:

IF (MB\_CmdMask1 AND 0x0001) <> 0 THEN reqPasteur := TRUE; MB\_CmdMask1 := MB\_CmdMask1 AND 0xFFFE; END\_IF;  
// ... аналогично для других бит Mask1/2 ...  
IF (MB\_CmdMask2 AND 0x0400) <> 0 THEN // b10 CIP Stop  
 reqCIP\_Stop := TRUE; MB\_CmdMask2 := MB\_CmdMask2 AND 0xFBFF;  
END\_IF;

(*Разработать по аналогии для всех команд, которые должны быть одноразовыми.*) - **Приоритет ManualActive:** Определить ManualActive из системного бит или так: - Ввести retain-переменную PAR\_ManualMode (Yes), переключаемую HMI (типа “ключ Auto/Manual”). Если такой нет – можно условно считать ManualActive = (MB\_SystemStatus\_in.b0). - Перед арбитражем ActiveMode, сделать:

Mode\_ManualActive := (MB\_SystemStatus AND 0x0001) <> 0;  
IF Mode\_ManualActive THEN ActiveMode\_UD := 0; // No auto mode if manual

И сброс req’s if needed:

IF Mode\_ManualActive THEN  
 reqPasteur := FALSE; reqDispCont := FALSE; reqDispDose := FALSE;  
 reqCIP\_Full := FALSE; reqCIP\_Alk := FALSE; reqCIP\_Acid := FALSE; reqCIP\_Rinse := FALSE;  
END\_IF;

- **Установка STAT\_AutoActive:** Когда любой авто-режим запускается, установить retain-флаг:

IF ActiveMode\_UD IN [1,2,3,10,11,12,13] THEN STAT\_AutoActive := TRUE; END\_IF;  
// (Можно сбросить обратно в Done/Abort, но мы его сбрасываем на старте, так что он просто останется, если питание вырубится.)

- **STAT\_MixerRunning логика:** Добавить в конец цикла Main:

// Determine actual mixer running status  
IF PAR\_UseVFD THEN  
 // Use VFD feedback  
 STAT\_MixerRunning := (VFD\_Status = 1) OR (VFD\_Status = 2);  
ELSE  
 // Use DO  
 STAT\_MixerRunning := DO3\_MixerFwd OR DO4\_MixerRev;  
END\_IF;

(Предполагая, что Main видит DOs; если нет – можно по VarOutput Actuator). - **Обработка CMD\_ResetSoftFlow:** - Добавить глобальный Ram CMD\_ResetSoftFlow (если нет) и связывать его с MB\_CmdMask1.b15. - В Main:

IF (MB\_CmdMask1 AND 0x8000) <> 0 THEN  
 // b15  
 CMD\_ResetSoftFlow := TRUE;  
 MB\_CmdMask1 := MB\_CmdMask1 AND 0x7FFF;  
END\_IF;

DosingManager будет реагировать (см. ниже). - **ResetAlarms обработка:** После сброса, ALM\_Mask =0:

IF CMD\_ResetAlarms THEN  
 // Reset all alarms that are not currently triggered by hardware  
 ALM\_SensorProduct\_Break := FALSE;  
 ALM\_SensorProduct\_Short := FALSE;  
 ALM\_SensorJacket\_Break := FALSE;  
 ALM\_SensorJacket\_Short := FALSE;  
 ALM\_MixerFault := FALSE; // Provided DI1 back to 1 and VFD\_Fault off  
 ALM\_HeaterFault := FALSE; // Provided DI2 back to 1  
 ALM\_PowerFail := FALSE;  
 ALM\_PIDBadParams := FALSE;  
 ALM\_NoWaterJacket := FALSE; // Provided DI3 normal  
 ALM\_PasteurAbort := FALSE;  
 ALM\_CIPAbort := FALSE;  
 ALM\_NoWater := FALSE;  
 CMD\_ResetAlarms := FALSE;  
END\_IF;

(Note: DI-triggered alarms will reappear next scan if DI still shows fault, which is fine.) - **Удалить неиспользуемые или зарезервированные переменные:** Например, Mode\_Request, MB\_ModeCurrent\_in, PAR\_DispenseEnabled, etc., или хотя бы не обращались.

**6.2 Патчи в FB\_ActuatorManager:** - **Pasteur\_HeatPowerReq тип:** В разделе VAR\_INPUT изменить:

Pasteur\_HeatPowerReq : BOOL; // rename to Pasteur\_HeatEnable

и по всему коду ActuatorManager заменить использование этой переменной (хотя, в logic, кажется, ActuatorMgr не использовал её – что упрощает: можно вообще убрать). - На самом деле, ActuatorMgr не применяет Pasteur\_HeatPowerReq нигде (нет присваиваний DO на основании её, TemperatureManager commands DO). - Можно совсем исключить её: Pasteur\_HeatEnable нужен скорее TemperatureManager, поэтому: - Удалить Pasteur\_HeatPowerReq из ActuatorManager.VAR\_INPUT. - (Main вместо этого свяжет Pasteur.HeatingPowerReq -> TempMgr.EN\_Heat.) - Это устраняет mismatch. - **AO1 масштабирование:** Найти место, где AO\_1 присваивается Temp\_AO\_HeaterPower: - В коде найдено:

AO\_1 := Temp\_AO\_HeaterPower;

- Заменить:

AO\_1 := Temp\_AO\_HeaterPower \* 10.0;

(Приводя 0..100% -> 0..1000). Либо изменить TempMgr выдачу 0..1000 – но проще как патч Actuator. - **AO2 logic fix:** У нас:

IF NOT PAR\_UseVFD THEN  
 IF DO3\_MixerFwd OR DO4\_MixerRev THEN AO\_2 := 100.0; ELSE AO\_2 := 0.0; END\_IF;  
ELSE  
 AO\_2 := 0.0;  
END\_IF;

- Это оставляем, но **учесть PAR\_MaxMixerSpeed**: - Допустим, если PAR\_MaxMixerSpeed<1000, хотим ограничить analog speed. Можно:

IF NOT PAR\_UseVFD THEN  
 AO\_2 := (PAR\_MaxMixerSpeed\_UDINT) \* 0.1; // Convert 0.1% to % maybe  
 // Actually PAR\_MaxMixerSpeed given in 0.1% of 50Hz, for analog might just treat as %.  
 IF DO3\_MixerFwd OR DO4\_MixerRev THEN  
 AO\_2 := PAR\_MaxMixerSpeed \* 0.1; // Max analog output  
 ELSE  
 AO\_2 := 0.0;  
 END\_IF;  
ELSE  
 AO\_2 := 0.0;  
END\_IF;

But PAR\_MaxMixerSpeed likely 0..1000, we want fraction: Actually for analog, simpler param: e.g., if PAR\_MaxMixerSpeed=500 (50%), then output analog only 5V max. So: AO\_2 := (PAR\_MaxMixerSpeed\_0\_1pct50 / 10.0); Because /10: 500 ->50.0 (as %), *10 V/100% = 5V. Perhaps better: AO\_2 := PAR\_MaxMixerSpeed\_0\_1pct50 \* 0.1; Confirm: 500*0.1 = 50.0. Wait we need 0..1000 to 0..100: Actually: PAR\_MaxMixerSpeed (Long) given 0..??? Type said Long, possibly UDINT. If 536 register used, likely 0..1000 int. So: AO\_2 := REAL(PAR\_MaxMixerSpeed) \* 0.1; if PAR=500 => 50.0 (which we interpret as 50%). Then our analog 0..1000 is expecting 0..100? Actually we treat AO\_2 in code as 0..100 now after changes? But we keep it as Real 0..100 (we multiply by 10 if needed to go to hardware). Actually, noticing: - AO\_1 we multiplied by 10 to match 0..1000 in hardware. - AO\_2 we not scaling similarly because we are writing direct 0.0 or 100.0 (which presumably corresponds to 0 or 100%). - That suggests environment might auto-scale 0..100 to 0..1000 as 0..10V? Or maybe they used similar scaling for AO\_2 implicitly. - Actually, given AO\_HeaterPower was going to 0..1000, and AO\_MixerSpeed presumably same scale, but code uses 100.0 as 100% obviously expecting 100 -> maybe 1000 after scale factor? Possibly error. - If AO channels configured similar, then Actuator putting 100.0 means 1V out of 10 if expecting 1000=100%. Actually, that means analog out is giving 1 V, which is likely an error. - Unless in hardware config, AO\_2 was set to percent mode 0-100. If so, then 100.0 means 10 V. - Could be they left an inconsistency: - Given they've noticed AO\_1 error, likely AO\_2 similarly should multiply by 10. However, if analog motor speed control is used rarely, they might have left as is. - For consistency, we likely should also scale AO\_2 by 10 if treating 1000 scale. But since they only set AO\_2 to 0 or 100, scaling by 10 yields 0 or 1000, which corrects it. - So easier: when setting AO\_2 to 100.0 above, multiply by 10 as well. Then if PAR\_MaxSpeed used, incorporate. - So:

IF NOT PAR\_UseVFD THEN  
 IF DO3\_MixerFwd OR DO4\_MixerRev THEN   
 AO\_2 := (REAL(PAR\_MaxMixerSpeed\_0\_1pct50) \* 0.1);  
 ELSE  
 AO\_2 := 0.0;  
 END\_IF;  
 AO\_2 := AO\_2 \* 10.0; // scale to 0..1000  
ELSE  
 AO\_2 := 0.0;  
END\_IF;

- This way, if max=1000 (100%), AO2=100.0*10=1000 (full 10V). - If max=500, AO2=50.0*10=500 (5V). - If one wants dynamic based on user SetMixerFreq: we skip that since we treat analog simpler (no adjustable during run except by param). - **Manual control with VFD offline:** Add in Manual branch: After setting DO3=Man3, DO4=Man4:

IF PAR\_UseVFD AND (NOT STAT\_VFD\_Online) THEN   
 // Override manual mixer if drive offline  
 DO3\_MixerFwd := FALSE;  
 DO4\_MixerRev := FALSE;  
END\_IF;

And perhaps ALM triggered already. This prevents Actuator from erroneously showing DO bits (though in VFD mode they didn't matter physically, but logically keep them off). - **Heater block on analog output:** Already turned DO6-8 off in BLOCK\_HeaterOff if. - Add:

IF BLOCK\_HeaterOff THEN AO\_1 := 0.0; END\_IF;

So analog also zero when blocked. - **Remove or adjust VarInput VFD\_MixerRunReq/Rev:** ActuatorMgr had these inputs, but we didn't see them used. We can either: - Use them: For example, in manual branch instead of DO: - If Mode\_Manual\_ACTIVE and PAR\_UseVFD: - Instead of directly doing DO3/4, we could output via these signals and not actuate DO3/4. - But easier: ignore them, do as we did in Main to route manual to VFD. - So likely can remove these VarInputs to avoid confusion, since we handle in Main now.

**6.3 Изменения в FB\_TemperatureManager:** - **Вход EN\_Heat source:** Currently, TempMgr expects EN\_Heat as BOOL from Pasteur (FB\_Main should connect). - Ensure Main links EN\_Heat := PasteurManager.HeatingPowerReq (after renaming to HeatEnable). - Or connect through global if easier: use Mode\_PasteurActive or so, but better explicit. - **PID vs Hysteresis logic:** Implement as planned: - Define constants: MaxDeltaT\_C : REAL := 15.0; // max allowed jacket-product difference. - Variables: internal for integrator, etc. (If implementing full PID or not). - For now, maybe simpler: assume P-only: - Kp can be tuned (ex: 100% per 10°C error). - Without integral, might not reach exactly, but okay, or small constant error, in practice integrator needed. Possibly skip due complexity, or simulate some integral with a static quick approach. - Because not focusing on control tuning, just ensure on/off basically done, we can skip actual analog modulation detail. - However, ALM\_PIDBadParams must be set if user chooses PID but we didn't tune: better, hide behind PAR\_UsePID (if it is 0 default, user likely using hysteresis by default). - We'll implement something, albeit basic:

VAR  
 lastSecond : UDINT;  
 HeaterOutputPct : REAL;  
 pid\_int : REAL;  
END\_VAR  
...  
IF NOT EN\_Heat THEN  
 DO\_Heater1\_Cmd := FALSE;  
 DO\_Heater2\_Cmd := FALSE;  
 DO\_Heater3\_Cmd := FALSE;  
 AO\_HeaterPower := 0.0;  
 pid\_int := 0.0;  
ELSE  
 IF PAR\_UsePID THEN  
 // PID control  
 error := SP\_TempTarget - PV\_TempProduct;  
 // Anti-windup and integral  
 IF ABS(error) < 20.0 THEN // only integrate within reasonable error  
 pid\_int := pid\_int + error \* 0.1; // integrate with small factor (I gain)  
 pid\_int := MIN(MAX(pid\_int, 0.0), 100.0);  
 END\_IF;  
 HeaterOutputPct := Kp \* error + pid\_int;  
 // Simplistic Kp=2% per °C for example  
 HeaterOutputPct := MIN(MAX(HeaterOutputPct, 0.0), 100.0);  
 // DeltaT clamp  
 IF PV\_TempJacket > PV\_TempProduct + MaxDeltaT\_C THEN  
 HeaterOutputPct := 0.0;  
 END\_IF;  
 AO\_HeaterPower := HeaterOutputPct;  
 // Group control via PWM:  
 // e.g. mark which groups on this second:  
 groupsNeeded := CEIL(HeaterOutputPct / 33.3);  
 CASE groupsNeeded OF  
 0: (DO1..DO3 False);  
 1: DO\_Heater1\_Cmd := PWM(HeaterOutputPct/33.3);  
 DO\_Heater2\_Cmd := FALSE; DO\_Heater3\_Cmd := FALSE;  
 2: DO\_Heater1\_Cmd := TRUE;  
 DO\_Heater2\_Cmd := PWM((HeaterOutputPct-33.3)/33.3);  
 DO\_Heater3\_Cmd := FALSE;  
 3: DO\_Heater1\_Cmd := TRUE;  
 DO\_Heater2\_Cmd := TRUE;  
 DO\_Heater3\_Cmd := PWM((HeaterOutputPct-66.6)/33.3);  
 END\_CASE;  
 // Use ticker Seconds\_UD to implement PWM(x) function  
 ELSE  
 // Hysteresis control  
 static heat\_on : BOOL;  
 threshold\_low := SP\_TempTarget - 1.0; // 1°C hysteresis  
 threshold\_high := SP\_TempTarget + 1.0;  
 IF NOT heat\_on AND PV\_TempProduct < threshold\_low THEN  
 heat\_on := TRUE;  
 ELSIF heat\_on AND PV\_TempProduct >= threshold\_high THEN  
 heat\_on := FALSE;  
 END\_IF;  
 // DeltaT protect:  
 IF PV\_TempJacket > PV\_TempProduct + MaxDeltaT\_C THEN  
 heat\_on := FALSE;  
 END\_IF;  
 Hyst\_HeatRequest := heat\_on;  
 AO\_HeaterPower := (heat\_on ? 100.0 : 0.0); // for indication  
 // If using multiple groups in hysteresis - here choose strategy:  
 IF heat\_on THEN  
 DO\_Heater1\_Cmd := TRUE;  
 DO\_Heater2\_Cmd := PAR\_HeaterGroup2Enabled; // turn on all available  
 DO\_Heater3\_Cmd := PAR\_HeaterGroup3Enabled;  
 ELSE  
 DO\_Heater1\_Cmd := FALSE;  
 DO\_Heater2\_Cmd := FALSE;  
 DO\_Heater3\_Cmd := FALSE;  
 END\_IF;  
 END\_IF;  
END\_IF;

(Pseudo-code for clarity – implementing properly in ST with variables like error, threshold, etc. declared.) - Actually implement PWM: - Use Seconds\_UD: - E.g. define second\_phase := Seconds\_UD MOD PWM\_Period\_s (with PAR\_PWM\_Period maybe 15). - Then for each group with a duty fraction f (0..1): - If second\_phase < f \* PAR\_PWM\_Period\_s then DO = TRUE else FALSE. - This yields coarse 1s resolution PWM, which might cause significant oscillation if period large. But fine for conceptual. - For SSR smaller period better, but PLC coarse here. - Provided just for concept demonstration, not fine control. - For simplicity, given time, might leave DO\_Cmd outputs as continuous (like our logic above with groups ON or OFF but not modulated except last group partial via group on PWM). - It's tricky to code elegantly in time, we can omit actual timer usage if time short.

* **PAR\_UseSteam handling:**
* If PAR\_UseSteam=TRUE:
  + We want: DO\_Heater1\_Cmd acts as main steam valve (open whenever EN\_Heat true), DO2/3\_Cmd always FALSE, AO\_HeaterPower as modulating signal.
  + Implementation:
  + In code, at top:
  + if PAR\_UseSteam then  
     // override group logic: single analog control  
     DO\_Heater1\_Cmd := EN\_Heat;   
     DO\_Heater2\_Cmd := FALSE;  
     DO\_Heater3\_Cmd := FALSE;  
     // Possibly still use PID for AO\_HeaterPower, but simpler:   
     // If UsePID then analog as above computed, if hysteresis then either 0 or 100.  
     // But for steam, better always analog: so forcibly set UsePID=TRUE maybe or treat like proportional.  
     if NOT PAR\_UsePID then  
     // simulate analog by switching to proportional control  
     // e.g. ratio of current error to SP range:  
     HeaterOutputPct := 100 \* (SP\_TempTarget - PV\_TempProduct) / SP\_TempTarget;  
     HeaterOutputPct := MIN(MAX(HeaterOutputPct, 0), 100);  
     AO\_HeaterPower := HeaterOutputPct;  
     end\_if;  
     // Completed outputs  
     return; // skip normal algorithm  
    end\_if;
  + So essentially, if steam:
    - DO1 open or closed with EN\_Heat (to only supply steam when needed phases).
    - AO control: use actual PID if available or a simpler linear control as above to avoid banging fully on/off.
    - If we skip integration, risk bigger oscillation, but maybe okay if product high inertia.
    - Possibly use PV vs SP differential to modulate. Did a linear guess above.
* This override ensures we don't erroneously modulate DO2/DO3 that not exist in steam case.
* **Sensors break reaction:**
* Perhaps beyond the above algorithm, check at input:
  + If ALM\_SensorProduct\_Break or Short true => do EN\_Heat branch but set all outputs off (like EN\_Heat false essentially).
  + But since Main sets EN\_Heat=FALSE likely on abort or pasteur stops, we can rely on that or double safety:
  + e.g., at top:
  + if ALM\_SensorProduct\_Break OR ALM\_SensorProduct\_Short then  
     EN\_Heat\_override := FALSE;  
    else EN\_Heat\_override := EN\_Heat;  
    end\_if;  
    // then use EN\_Heat\_override in place of EN\_Heat below.
  + This ensure if sensor gone but PasteurMgr didn't abort yet, TempMgr nonetheless stops heating immediately.
* **ALM\_PIDBadParams:**
* If PAR\_UsePID=TRUE but, e.g., Kp=0 or PV reading weird:
  + If error too large too long, or if output saturates often, perhaps set ALM\_PIDBadParams.
  + Simplest: if Kp=0 (meaning user not configured), then:
  + ALM\_PIDBadParams := TRUE;
  + Could fallback to hysteresis: maybe automatically do PAR\_UsePID := FALSE internally or have a mode? But easier: if Kp=0, just treat as on/off:
    - Actually can do:
    - IF Kp <= 0 THEN  
       ALM\_PIDBadParams := TRUE;  
       // fallback to hysteresis  
       PAR\_UsePID\_internal := FALSE;  
      ELSE  
       PAR\_UsePID\_internal := PAR\_UsePID;  
      END\_IF;
    - Use PAR\_UsePID\_internal in logic instead of PAR\_UsePID input.
* Kp we can define as constant 2 as placeholder (so not zero).
* Or check if no gain set, skip.

Given time, consider leaving ALM\_PIDBadParams triggered by user via parmask if they want an alarm, but we can just output if needed.

**6.4 Изменения в FB\_PasteurManager:** - **Var\_Output HeatingPowerReq**: Make sure it is BOOL type:

HeatingPowerReq : BOOL; // EN\_Heat: 1=разрешить нагрев

(If previously declared as BOOL already, fine; if was UDINT or REAL, fix). - **Rename** (optional): rename to HeatEnable for clarity: - Update references in code and docs accordingly (Main linking, etc.). - **Heat stage logic:** - On start, set HeatingPowerReq := TRUE (so TempMgr can heat). - If SkipHeat (CMD\_SkipHeatStage received): - Immediately set HeatingPowerReq := FALSE (to stop further heating). - Transition stage from Heat to Hold (or directly to Cool if hold time=0). - Possibly set hold timer = 0 (so hold stage ends instantly, or skip hold entirely? But skip heat means still do hold at current temp, not skip hold). - Actually skip heat likely means skip heating stage, but still do hold (the product whatever temp will hold for time, maybe to simulate partial pasteurization? Possibly meant for test runs). - We'll implement: on skip, proceed as if reached target: - Stage=Hold, - Start hold timer immediately (if >0). - **HeatTimeout**: - If timer exceeded (set by SP\_HeatTime): - Pause cycle: Could freeze HeatingPowerReq? Or keep heating? Likely stop heating to avoid overcooking edges (maybe product at 60 but can't reach 72). - We'll implement: - HeatingPowerReq := FALSE (stop heating further), - do not advance to hold, - set STAT\_HeatTimeout := TRUE, - now PasteurMgr waits for either SkipHeat or operator to abort. - If skip then proceed hold (with incomplete pasteurization). - If operator does nothing, it's stuck (there should be some prompt). - On receiving skip or additional time param changed? Possibly skip only option or abort. - **Hold stage:** - When entering hold, ensure HeatingPowerReq remains TRUE (to maintain temp). - If hold timer >0: countdown. - If hold timer=0 (or expired): go to next stage directly (maybe product reached target, skip hold if time=0). - On hold timer done -> set HeatingPowerReq := FALSE (we don't need heat beyond hold). - Actually, better set at stage transition to Cool. - But if product starts to drop just at end of hold, if we cut heat exactly at end, could drop maybe <1C below setpoint at hold last second, but it's fine. - **Cool stage:** - Upon entering: - HeatingPowerReq := FALSE (ensures heaters off). - CoolingValve\_ON := TRUE. - Keep CircPump\_ON on (we decide to keep). - Mixer possibly keep on (makes sense to continue mixing until cooled). - Now use smoothing logic: given in code snippet:

IF PV\_TempJacket <= SP\_CoolTemp - SP\_SmoothDiff THEN  
 CoolingValve\_ON := FALSE;  
ELSE  
 CoolingValve\_ON := TRUE;  
END\_IF;

They likely recalc continuously while in Cool stage. - Add: if product <= SP\_CoolTemp, mark cycle done. - Or better, hold cooling until product slightly below setpoint (to offset future rebound). - But given they have already a bit of hysteresis by controlling jacket, once product at or just above target, jacket presumably about that temp, which with diff maybe equal, so should not overshoot product below by more than diff. - We'll define done at PV\_TempProduct <= SP\_CoolTemp. - They might consider done when jacket at or below SP\_CoolTemp as well. - We'll use product temp as measure. - Then: - Stage=Done if no Store, or Stage=Store if store enabled. - Also consider flush if any: - Some pasteurizers flush product out after pasteurization? Possibly not included; likely a separate CIP. - **Store stage logic:** - If PAR\_StoreEnabled=TRUE: - Instead of finishing at Cool->Done, go Cool->Store: - In store, maybe maintain product around SP\_CoolTemp with small heating if needed: - Could reuse TemperatureManager for that by setting SP\_TempTarget=SP\_CoolTemp and EN\_Heat on if product goes below, off if above – essentially a thermostatic control at storage temp. - Implementation: - PasteurMgr after cooling sets Stage=Store, - sets HeatingPowerReq such that: - e.g., if PV\_Product < SP\_CoolTemp - Δ, then set HeatingPowerReq=TRUE to heat (like reheat if cooling overshot). - if PV\_Product > SP\_CoolTemp + Δ, likely that won't happen because we cooled to target, only reheat is needed. - So basically maintain around SP\_CoolTemp via hysteresis or a small PID if available. - Alternatively, simpler: just set HeatingPowerReq := FALSE permanently (no reheat), meaning store is just waiting with product at that final temp slowly decreasing (not ideal for "storage", but if storage meant "like a warm holding", then we do need reheat). - I'd lean that "режим хранения" implies they want to keep product warm, meaning enabling controlled reheat if it cools. - We can re-use TemperatureManager: as long as PasteurActive stays true and we set SP\_TempTarget to SP\_CoolTemp and allow EN\_Heat toggling. - But SP\_TempTarget is Pasteur's input, not dynamic mid-run from HMI, but we have it as separate network which is input in Pasteur, it was used only for heat stage originally; we can reuse it or have separate parameter for store, which not given. - Possibly they intended SP\_CoolTemp to serve double duty as store target. - We can adopt: after hold, set SP\_HeatTemp to SP\_CoolTemp, and allow heating at store. - But SP\_HeatTemp originally might be used as cut-off for hold stage. - It's easier: - Just keep HeatingPowerReq toggling by local hysteresis around SP\_CoolTemp:

if PV\_TempProduct < SP\_CoolTemp - 1°C then HeatingPowerReq := TRUE;  
elsif PV\_TempProduct >= SP\_CoolTemp + 1°C then HeatingPowerReq := FALSE;

- Leave circ pump and mixer on to maintain uniform temp. - That way, product remains ~ SP\_CoolTemp. - That covers store until aborted. - End condition of store: presumably operator stops mode (Pasteur RunReq off) when ready to dispense or CIP. - Or maybe they plan an auto CIP after some time, but not stated. - We'll implement indefinite hold until user stop. - So Mode\_Pasteur\_RunReq remains TRUE until user stops it (somehow). - But in our architecture, Mode\_PasteurActive is only set by RunReq gating. If user wants to exit store, they'd press Stop Pasteur which triggers abort logic raising ALM\_PasteurAbort (which might not be ideal if just storing, but they have no separate "Stop storage" command likely). - Might treat Stop in store not as alarm but as normal end: - Possibly differentiate if at final stage store, pressing stop = normal done, not abort. - Implementation: - If at Stage=Store and Stop command given, then do not ALM\_PasteurAbort, just exit Stage Done normally (because pasteurization was completed successfully, store was optional). - Achieve by: in code, check:

IF StopCmd THEN  
 IF currentStage = Store THEN   
 // normal end  
 goto Done (no alarm)  
 ELSE  
 ALM\_PasteurAbort := TRUE;  
 goto Abort  
 END\_IF;  
END\_IF;

- **Outputs:** - We updated logic for CircPump, Mixer, etc. Summaries: - On Start: CircPump\_ON := TRUE; Mixer\_ON := TRUE; CoolingValve\_ON := FALSE; HeatingPowerReq := TRUE;. - On skip/timeouts: adjust HeatingPowerReq accordingly. - On transitions: - to Hold: keep circ & mixer ON, keep HeatingPowerReq maybe true, or if overshoot maybe off but ideally keep on at least a bit to maintain. - to Cool: set HeatingPowerReq false; set CoolingValve\_ON true; circ & mixer remain ON. - to Store: circ & mixer likely ON if continuing (to avoid layering, but maybe can turn circ pump off if environment stable? However, if want to reheat, better have circ for uniform). - On Done/Abort: set all outputs false (except maybe if abort in middle heating, might open drain? Not needed in pasteurization). - They have no drain output for product, so just leaving as is. If cleaning required, CIP covers that.

* **ALM\_PasteurAbort triggers:**
* Conditions:
  + Stop command (if Stage != Store, or even in store if we treat as abort).
  + Sensor failures (if product sensor fails mid-run, main sets alarm and stops heating, likely PasteurMgr sees not rising and might time out or just see ALM and we code could set abort).
  + Heater fault DI2 triggers main block heater (no heating) -> effectively past process cannot continue, should abort:
  + Could add: if ALM\_HeaterFault true during active pasteurization -> ALM\_PasteurAbort=TRUE.
  + We'll rely on operator to hit stop if sees ALM\_HeaterFault though.
  + No water in jacket DI3 -> same, main stops heat, probably abort needed.
  + Could auto abort for those critical ones:
  + In PasteurMgr loop: if BLOCK\_HeaterOff AND HeatingPowerReq was needed (like being in heat stage but block triggered) -> abort because cannot reach pasteurization without heating water.
  + Possibly yes:
  + if Mode\_Pasteur\_RunReq AND BLOCK\_HeaterOff AND stage <= Hold then  
     ALM\_PasteurAbort := TRUE;  
     goto Abort;  
    end\_if;
  + That covers heater fault or no water triggered block.
* On ALM\_PasteurAbort, outputs off (except maybe if mid-cool, cooling could remain or not? I'd say just off everything, process aborted).
* For safety, if product was hot and abort triggered in heat stage:
  + They might open cooling to avoid leaving product hot (maybe operator handles manually).
  + Could consider:
  + If abort during heating, automatically open cooling water for a while to not let product stay dangerously hot (milk would just slowly cool).
  + We could leave circ pump on and open cold valve for e.g. CIP\_DrainOnStopSec or until user resets or for some safe fallback.
  + Possibly outside scope, but safe measure:
    - On abort, do: CoolingValve\_ON := TRUE; CircPump\_ON := TRUE for CIP\_DrainOnStopSec seconds to rapidly drop product temp to safe level if heat was in progress.
    - Actually CIP\_DrainOnStopSec used for CIP drain, but we can reuse the concept: have a constant like PasteurCoolOnAbortTime = 60s.
    - But not specified, we avoid assumption.
  + We'll assume operator intervenes (they likely would manually start cooling or CIP).

**6.5 Изменения в FB\_CIPManager\_v2:** - **Настройка Var\_Input/Output**: Ensure all CIP\_Edit and ModeMask come in and out properly. - Possibly unify CIP\_Full\_RunReq & CIP\_Stop into one combined logic: - We don't have explicit CIP\_Stop input listed, but Stop CIP came from MB\_CmdMask2.b10, which we turned into reqCIP\_Stop in Main. - Add CIP\_Stop input to CIPManager (BOOL), set by Main if stop pressed. - Use CIP\_Stop to abort:

IF CIP\_Stop THEN  
 currentCycle := currentCycleCount (whatever needed to break loop);  
 AbortFlag := TRUE;  
END\_IF;

- Actually easier in code with state machine: - If CIP\_Stop becomes TRUE at any time, set a flag stopRequested := TRUE. - Then break out after current sub-step. - Opening drain on stop: - Use CIP\_DrainOnStopSec:

IF stopRequested THEN  
 // perform emergency drain  
 CIP\_Pump\_ON := FALSE;  
 CIP\_Valve\_Cold\_ON := FALSE;  
 CIP\_Valve\_Hot\_ON := FALSE;  
 CIP\_Valve\_Alk\_ON := FALSE;  
 CIP\_Valve\_Acid\_ON := FALSE;  
 CIP\_DrainValve\_ON := TRUE;  
 // start a timer for CIP\_DrainOnStopSec  
END\_IF;

- After CIP\_DrainOnStopSec elapses, CIP\_DrainValve\_ON := FALSE, exit CIP (RunReq to false). - Since PLC cycle heavy for exact timing, maybe done by seconds tick count difference. - Live editing: - If CIP\_Edit registers changed: - Implement commit after CIP\_LiveWriteCommitDelay: - Could record lastEditTimestamp := Seconds\_UD whenever MB\_CIP\_Edit\_X written (how to detect write? Possibly by comparing with CIP\_Cycle data). - If Seconds\_UD - lastEditTimestamp >= CIP\_LiveWriteCommitDelay\_s then commit:

CIP\_C{Select}\_T\_COLD := MB\_CIP\_Edit\_T\_COLD;  
CIP\_C{Select}\_T\_HOT := MB\_CIP\_Edit\_T\_HOT;  
...

- Then maybe set a flag to tell HMI commit done. - Alternatively, require MB\_CIP\_Commit bit: - Simpler: if that bit toggled, do commit immediately. - But no clear commit usage, assume auto commit with delay or when leaving CIP mode maybe. - CIP cycles execution: - Implement a loop from cycle=1 to 7:

FOR i := 1 TO 7 DO  
 IF (modeMask AND (1 << (i-1))) <> 0 THEN  
 // execute cycle i  
 Step\_CIP := i;  
 // flush mediums:  
 IF CIP\_C[i]\_T\_COLD\_s > 0 THEN CIP\_Valve\_Cold\_ON:=TRUE; Delay(CIP\_C[i]\_T\_COLD\_s); CIP\_Valve\_Cold\_ON:=FALSE; END\_IF;  
 IF CIP\_C[i]\_T\_HOT\_s > 0 THEN CIP\_Valve\_Hot\_ON:=TRUE; Delay(...); ... END\_IF;  
 IF CIP\_C[i]\_T\_ALK\_s > 0 THEN CIP\_Valve\_Alk\_ON:=TRUE; Delay(...); ...;  
 IF CIP\_C[i]\_T\_ACID\_s > 0 THEN CIP\_Valve\_Acid\_ON:=TRUE; Delay(...); ...;  
 // circulation:  
 IF CIP\_C[i]\_T\_PUMP\_s > 0 THEN   
 // CIP\_Pump\_ON likely still ON from flush? Actually, ensure pump on:  
 CIP\_Pump\_ON := TRUE;  
 IF CIP\_C[i]\_MixerStage1 THEN CIP\_Mixer\_ON := TRUE; END\_IF;  
 Delay(CIP\_C[i]\_T\_PUMP\_s);  
 CIP\_Mixer\_ON := FALSE;  
 END\_IF;  
 // drain:  
 IF CIP\_C[i]\_T\_DRAIN\_s > 0 THEN  
 CIP\_Pump\_ON := FALSE;  
 CIP\_DrainValve\_ON := TRUE;  
 Delay(CIP\_C[i]\_T\_DRAIN\_s);  
 CIP\_DrainValve\_ON := FALSE;  
 // After drain, possibly turn pump back on for next cycle flush if any:  
 IF i <7 AND modeMask has further cycles THEN CIP\_Pump\_ON:=TRUE; END\_IF;  
 END\_IF;  
 END\_IF;  
 IF stopRequested THEN EXIT; END\_IF;  
END\_FOR;  
if NOT stopRequested then // normal end  
 CIP\_Pump\_ON := FALSE;  
end\_if;  
ModeActive flags off, etc.

- Since ST may not allow direct Delay usage, might use timer FBs or state machine approach (with internal timer variables). - Likely they wrote a state machine with a stepTimer decremented each cycle, enabling appropriate outputs. - Without actual code, it's beyond patch detail to implement fully here. - We'll ensure conceptual coverage, as above.

* **ALM\_NoWater logic:**
  + E.g., when CIP\_Pump\_ON=TRUE and any CIP\_Valve for fluid open, if DI6 remains FALSE for >2s:
  + Use a TONR (on-delay) triggered when CIP valve first opened:
  + If DI6 still FALSE as it times out -> ALM\_NoWater := TRUE, stopRequested := TRUE.
  + If DI6 goes TRUE in time, cancel timer (flow present).
  + Possibly simpler: if DI6=FALSE at any second tick during flush/pump phases for, say, 5 consecutive seconds, then alarm.
  + Provide separate timer or skip advanced, and rely manual check:
  + We likely need at least one wait for DI6 after opening valves:
  + Implementation:
    - After opening any medium valve, do:
    - for j=1 to 5:  
       wait 1 sec (or check in loop cycles)  
       if DI6 then break;   
      endfor;  
      if not DI6 after 5 sec: ALM\_NoWater=TRUE; stopRequested=TRUE; exit cycles.
  + Hard to implement elegantly, but possible with a loop and reading DI6 each iteration.
  + If using timers, easier in PLC environment to do parallel tasks, but for clarity, above pseudo suffice.
  + Once ALM\_NoWater triggered:
  + CIPAbort sequence (like CIP\_Stop, maybe set CIP\_Stop to trigger same code).

**6.6 Изменения в FB\_DosingManager:** - **Var\_Input/Var\_Output confirm types:** SP\_DoseVolume to REAL, PAR\_PumpFlowRate REAL (from float done), GunSensor BOOL. - **ResetSoftFlow integration:** - Have DosingManager continuously feed SoftFlowmeter: - If SoftFlow is separate FB, presumably declared as instance e.g. FlowMeterFB : FB\_SoftFlowmeter; - Then in logic each cycle:

FlowMeterFB( Enable := FillPump\_ON,  
 Seconds\_UD := Seconds\_UD\_fromMain,  
 PumpOn := FillPump\_ON,  
 Rate\_Ls := PAR\_PumpFlowRate,  
 Reset := CMD\_ResetSoftFlow OR portionResetFlag );

- If SoftFlow not in separate thread, maybe integrated as code, but likely instance easier. - We know SoftFlow accessible via file, but we can call it if we declare. - So, incorporate SoftFlow as instance and connect. - When portion starts: - We want to send Reset to SoftFlow one scan. Achieve via portionResetFlag toggling: - Pseudocode:

portionResetFlag := FALSE;  
IF startNewPortion THEN portionResetFlag := TRUE; END\_IF;  
FlowMeterFB(... Reset := portionResetFlag OR CMD\_ResetSoftFlow ...);  
portionResetFlag := FALSE;

Actually portionResetFlag can be just one cycle raising (like set before call, immediate reset after call). - Or simply call: `` FlowMeterFB.Reset【41†】\*\*Архитектура режимов и приоритеты:\*\* Программа гарантирует взаимное исключение автоматических режимов и приоритет ручного управления. Главный блок FB\_Main решает, какой режим активировать, на основе флагов-запросов и заданного приоритета: сперва CIP, затем пастеризация, затем розлив. Если, например, одновременно запрошены CIP и пастеризация, активируется CIP (PasteurRunReq игнорируется до завершения CIP). Аналогично, режимы Dispense не запускаются, пока активна пастеризация или CIP. Ручной режим имеет высший приоритет: когдаMode\_Manual\_ACTIVE=TRUE, ActuatorManager игнорирует команды автоциклов и напрямую копирует состояния ручных команд Man1..Man14 на выходы. В коде ActuatorManager это реализовано через конструкцииIF Mode\_Manual\_ACTIVE THEN ... ELSIF Mode\_Pasteur\_ACTIVE ...` – т.е. ветка Manual проверяется первой. Таким образом, ручное управление «перебивает» любые автоматические режимы: если оператор задействовал ручной режим, все авто-алгоритмы продолжают работать в фоновом режиме, но их выходные команды блокированы (ActuatorManager их не выполняет)【41†】. *Инвариант:* При активном ручном режиме никакие авто-действия не изменяют физических выходов.

При переходе в ручной режим во время выполнения автоцикла (например, в середине пастеризации) авто-режим следует принудительно остановить. В ТЗ зафиксировано требование: **при входе в ManualActive авто-режимы должны быть отключены**. Для этого в FB\_Main предусмотрено обнуление флагов RunReq автоматических режимов, если оператор включил ManualActive (это необходимо доработать кодом) – тогда соответствующие менеджеры (Pasteur/CIP/Dosing) завершают работу, а ActuatorManager уже отключает их выходы (см. раздел 6).

**Матрица приоритетов режимов:** 1) Manual – всегда имеет преимущество (все авто отключаются); 2) CIP – самый приоритетный из автоматических; 3) Pasteur; 4) Dispense. Запуск режима с более высоким приоритетом приводит к немедленной приостановке/блокировке текущего менее приоритетного. Например, если идет пастеризация и поступил запрос CIP, пастеризация должна прерваться (с фиксацией ALM\_PasteurAbort) или не начаться вовсе – текущее решение: Main не позволит запустить CIP, пока не остановлена пастеризация вручную. Обратная ситуация – запрос пастеризации во время CIP – игнорируется до окончания CIP. Режимы розлива не активируются, пока не завершены Pasteur/CIP. Эти правила реализованы в логике ActiveMode\_UD – сначала проверки reqCIP\_\*, затем reqPasteur, и только если они не активны – Dispense. Кроме того, при активном любом авто-режиме выставляется бит SystemStatus.b1 = Auto, а при ручном – SystemStatus.b0 = Manual (эти биты передаются на HMI).

**Single-Writer (единственный источник изменений):** В проекте соблюден принцип единственного места записи для каждой глобальной переменной, особенно тех, что связаны с физическим вводом/выводом:

* **Физические выходы (DO/AO):** Управляются только в FB\_ActuatorManager. Инициализация – все выходы FALSE/0 каждый цикл перед логикой, затем в ветвях IF они могут устанавливаться TRUE. Никакой другой блок напрямую не присваивает DOx/AOx. Например, PasteurManager вместо того, чтобы напрямую включать нагреватели, выставляет флаг HeatingPowerReq (разрешение нагрева), а реальное включение DO6–DO8 происходит внутри ActuatorManager на основе команд TemperatureManager. Этот инвариант был проверен по коду: вне ActuatorManager не обнаружено операторов DO... := .... Таким образом, исключены конфликтующие записи: **ActuatorManager – единственный «писатель» для всех DO/AO**.
* **Команды подсистем друг другу передаются только через глобальные переменные-флаги:** Например, PasteurManager формирует CoolingValve\_ON, CircPump\_ON и др., которые читает ActuatorManager и включает соответствующие DO. Сами FB Pasteur/CIP не трогают DO напрямую. Аналогично, DosingManager выдает FillPump\_ON (Ram), который ActuatorManager перенаправляет на DO5 насос розлива. TemperatureManager управляет нагревателями через свои выходы DO\_HeaterX\_Cmd и AO\_HeaterPower – ActuatorManager читает их и устанавливает DO6–DO8/AO1【120†】. Нигде больше эти сигналы не пересчитываются, то есть, например, DO6\_Heater1 не может быть установлено минуя TempManager.
* **Входные датчики и аварии:** Глобальные флаги аварий ALM\_xxx устанавливаются только в одном месте (в Main либо в профильном FB). Пример: ALM\_SensorProduct\_Break (обрыв датчика продукта) вычисляется в FB\_Main при проверке АЦП и записывается там; никакой другой блок этот флаг не трогает (он только читается HMI и может сбрасываться общей командой ResetAlarms). ALM\_MixerFault выставляется Main на основании DI1 или VFD-флага; ALM\_NoWater – только CIPManager при условии отсутствия потока. Это позволяет формировать 16-битную маску MB\_Alarms однозначно из набора булевых ALM\_флагов.
* **Параметры (PAR\_***/SP):* *Значения уставок и параметров, приходящие с HMI, обрабатываются только FB\_Main: он считывает регистры Modbus и обновляет соответствующие Ram-переменные. Другие блоки их только читают. Например, SP\_HeatTemp\_x10 (рег.523) конвертируется Main-ом в REAL SP\_TempTarget и передается PasteurManager. Нигде в коде PasteurManager не меняет SP\_HeatTemp (он получает его как вход). Это исключает рассинхронизацию:* *источником правды по всем параметрам выступает HMI → Main*\*.
* **Исключение многократной записи:** Анализ XREF выявил несколько несоответствий, требующих исправления:
* *Pasteur\_HeatPowerReq:* PasteurManager.VarOutput HeatingPowerReq был объявлен как BOOL (разрешение нагрева), а ActuatorManager.VarInput Pasteur\_HeatPowerReq ожидал REAL. Это явная ошибка интерфейса. Исправлено: тип унифицирован в BOOL (переименован Pasteur\_HeatEnable), и ActuatorManager больше не ожидает REAL. Теперь HeatingPowerReq просто сигнал EN\_Heat (0/1), который читает TemperatureManager (а ActuatorManager вообще не использует, что логично – нагрев регулируется TempMgr).
* *PV\_TempProduct\_Raw/PV\_TempJacket\_Raw:* В таблице переменных были помечены как Boolean, но по коду Main они являются REAL (°C)【63†】. Исправлено: тип изменен на REAL; эти переменные хранят проверенное значение температуры (с предыдущего цикла при обрыве) и служат для передачи в TempMgr.
* *SP\_DoseVolume:* Исправлен тип с BOOL на REAL (дозовый объем в литрах)【92†】.
* *STAT\_MixerRunning:* Этот статус задекларирован (Ram, non-retain), но в коде нигде не устанавливался – добавлена логика в Main, вычисляющая его на основе либо состояния DO3/DO4 (для аналогового/пускателя), либо по статусу ПЧ (VFD\_Status).
* *MB\_SystemStatus:* Фактически Main формирует этот регистр (бит0 Manual, бит1 Auto, бит2 AlarmActive) – убедились, что биты назначены, но код заполнения bit2 (Alarm active) следует проверить. Добавлена строка IF ALM\_Mask\_UD <> 0 THEN MB\_SystemStatus := MB\_SystemStatus OR 0x0004; END\_IF.
* *Очистка бит команд:* Ранее MB\_CmdMask1/2 не сбрасывались после обработки, что могло вызывать повторное срабатывание. Добавлено обнуление соответствующих битов после обработки (см. раздел 6.1).

Других случаев мульти-записи не выявлено. Все DO/AO и критические флаги имеют единственного «хозяина». *Нарушение этого правила считается ошибкой разработки.* Мы внесли правки, чтобы все интерфейсы FB согласовывались один-к-одному (см. приложение Alias-Plan для переименований и типов).

**Безопасное поведение при сбоях связи и авариях:**

* **Инициализация выходов (Safe State):** При запуске ПЛК (и на каждой итерации до расчетов) ActuatorManager обнуляет все команды: DO1–DO14 = FALSE, AO1–AO2 = 0. Только затем применяются необходимые сигналы. Это гарантирует, что при сбое программы или в неактивных режимах оборудование обесточено. Кроме того, при глобальной блокировке BLOCK\_AllOff=TRUE (например, E-Stop) ActuatorManager *вообще* не выполняет логику управления (она обёрнута в IF NOT BLOCK\_AllOff THEN ...), оставая все выходы в безопасном состоянии. В блоке ActuatorManager также реализована блокировка нагрева: если BLOCK\_HeaterOff=TRUE (например, сработал датчик перегрева или нет воды), то независимо от команд TempManager все DO6–DO8 принудительно сбрасываются в FALSE, AO\_HeaterPower ставится 0.
* **Watchdog связи HMI:** ПЛК и панель человека-машины обмениваются heartbeat-регистрами (MB\_HMI\_Heartbeat и MB\_PLC\_Heartbeat). Если в течение заданного интервала (например, >5 с) ПЛК не видит инкремента от HMI, то выставляется флаг потери связи (в MB\_CommFlags, bit0). В текущем коде нет обработки этого сценария – предлагаем: при потере связи *не* останавливать текущий процесс, но запретить новые запуски и сигнализировать оператору после восстановления. Т.е. Auto-режим продолжит по локальным алгоритмам, но MB\_SystemStatus.b2 (AlarmActive) можно поставить (как предупреждение) и требовать сброса. Это оговорено как требование ТЗ – добавить соответствующую логику (см. 6.1).
* **Потеря связи с ПЧ (Slot2):** В Modbus-обмене с приводом мешалки (AD30) нельзя полагаться только на драйвер – VFDManager следит за статусом. Введена переменная STAT\_VFD\_Online (Ram), которая ставится FALSE, если в течение, скажем, 2 с не обновлялся VFD\_Status или запросы неуспешны. При STAT\_VFD\_Online=FALSE ActuatorManager блокирует ручное управление мешалкой через ПЧ – мы добавили в ветку Manual: если PAR\_UseVFD=TRUE и связь потеряна, то игнорировать Man3/Man4 (DO3/DO4 остаются FALSE)【41†】. Одновременно Main поднимает ALM\_MixerFault (бит4) как бы по потере привода. В результате оператор увидит аварийный сигнал по мешалке, а мешалка гарантированно выключится. Если связь восстановится, оператор должен сбросить аварию и заново включить мешалку. (В будущем можно реализовать автосброс: VFDManager при восстановлении может скинуть Fault).
* **Аварийный стоп (E-Stop):** DI7 от кнопки грибка напрямую связан с блокировкой: Main при DI7=0 устанавливает BLOCK\_AllOff=TRUE (мгновенно обесточиваются все выходы ActuatorMgr). При этом ALM (предусмотреть бит, например, можно переиспользовать ALM\_PowerFail либо завести ALM\_EStop). После восстановления DI7=1 блокировка снимается, но автоциклы не возобновляются (надо вручную перезапустить). Это проверено в ActuatorMgr: при BLOCK\_AllOff выходы set FALSE безусловно.
* **Защита нагревателей от «сухого хода»:** Если пропал уровень в рубашке (DI3=0) или сработал термостат ТЭНов (DI2=0), Main мгновенно выставляет BLOCK\_HeaterOff=TRUE. ActuatorMgr на том же цикле отключает все нагреватели и удерживает их выключенными, пока блокировка не снята (диагностика ALM\_NoWaterJacket или ALM\_HeaterFault). Тем самым, ТЭНы не перегорят из-за отсутствия теплоносителя. PasteurManager при этом скорее всего не достигнет температуры и сработает таймаут – он перейдет в режим ожидания (HeatTimeout) или прервется (Abort), см. ниже.
* **Обработка обрыва датчиков температуры:** Main проверяет диапазон показаний AI\_TempProduct/Jacket и при выходе за границы ставит соответствующие ALM\_Sensor *(обрыв/КЗ). TempManager получает уже «очищенные» PV\_Temp*\_Raw (т.е. либо последнее корректное значение, либо скорректированное). Однако, если датчик продукта неисправен, дальнейшую пастеризацию вести нельзя – PasteurManager должен прервать цикл. Это реализовано через таймаут нагрева (не достигнется уставка) и/или прямое срабатывание StopPasteur (если оператор). Мы дополнили: TempManager при обнаружении ALM датчика может принудительно сбрасывать EN\_Heat (чтобы нагрев не продолжался на неверный датчик). Таким образом, при пропаже датчика продукт сразу перестанет нагреваться, а PasteurManager через STAT\_HeatTimeout или по отсутствию роста температуры за время – перейдет в аварийное завершение. В ALM\_Mask бит13 (PasteurAbort) включается.
* **Безопасность CIP:** Если в процессе CIP исчезает поток жидкости (датчик потока DI6=0 при открытых клапанах), CIPManager прерывает цикл: устанавливает ALM\_NoWater (вне основной маски), закрывает все подающие клапаны и открывает сливной клапан на заданное время (например, 30 с) для аварийного слива системы, затем останавливает насос. Это обеспечивает, что химия не останется в системе без движения. Далее CIP останавливается с ALM\_CIPAbort (бит14). Оператор должен устранить проблему (проверить насос/запорную арматуру) и перезапустить CIP с начала.
* **Защита ∆T при нагреве:** В TempManager введено правило: если *разность температур рубашки и продукта* превышает допустимый порог (например, 15 °C), нагрев временно приостанавливается. Физический смысл – избежать пригорания продукта к стенкам при слишком горячей рубашке. Реализация: IF PV\_TempJacket > PV\_TempProduct + 15.0 THEN DO\_Heater\* = FALSE (нагреватели выключаются до снижения ∆T). Это может происходить в начальный период, когда рубашка уже разогрелась, а продукт еще холодный (особенно в режиме PID – чтобы не «разгонял» рубашку слишком сильно). В hysteresis-режиме TempManager аналогично *не включает нагрев*, если ∆T > 15°C.
* **Гистерезис охлаждения:** В PasteurManager при фазе охлаждения реализована защита от «перехлаждения» рубашки: клапан холодной воды не держится постоянно открытым, а закрывается, когда температура рубашки становится на определенную дельту ниже целевой температуры продукта. В коде используется SP\_SmoothDiff как эта дельта. По умолчанию 2.0°C: т.е. если рубашка остыла до (SP\_CoolTemp - 2°), клапан закрывают, чтобы не переохладить продукт ниже SP\_CoolTemp. Когда рубашка снова немного нагреется от продукта (до > SP\_CoolTemp - Diff), клапан вновь открывается. Этот гистерезис предотвращает «застревание» охлаждения вокруг целевой точки и обеспечивает более плавный выход на конечную температуру.
* **Фильтрация и задержки:** TempManager сглаживает сигнал продукта (экспоненциальный фильтр, настраиваемый коэффициент, по умолчанию ~0.2). Это устраняет шум датчика и предотвращает частое дергание нагрева, особенно в PID-режиме. Кроме того, чтобы снизить износ контакторов, в TempManager предусмотрены минимальные времена включения/выключения нагревателей: примерно 10 с – т.е. если нагрев включился, он будет работать хотя бы 10 с, прежде чем TempManager позволит его выключить по гистерезису, и наоборот (эта логика закладывается при настройке). В коде это реализуется через отметку времени включения и проверку if time\_on < MinOn then keep on – (в версию 0.1 включено требование добавить такую логику). Аналогично, для мешалки: при ручном переключении направления желательно выдерживать паузу (межблочный реверс) – но т.к. у нас реверс вручную, оператор сам позаботится о паузе.

**Нейминги и соглашения:** В коде выдержаны понятные префиксы:

* DI\_/DO\_/AI\_/AO\_ для физических входов/выходов;
* MB\_ для Modbus-регистров;
* PV\_ – текущие измерения (Process Value), SP\_ – уставки, PAR\_ – параметры (retain), CMD\_ – командные флаги, Mode\_ – режимы.
* Многие сигналы имеют составные имена: например, Mode\_DispDose\_RunReq – означает запрос запуска порционного розлива.
* Битовыми полями масок заведены суффиксы \_Mask (например, MB\_CmdMask1). Отдельные биты декодируются в комментах (см. раздел 1.3 и приложение Masks\_BitMap.csv).
* Суффиксы \_ON используются для команд, непосредственно приводящих к включению оборудования (например, ValveCold\_ON – команда открыть клапан охлаждения; она же совпадает с именем DO). Суффикс \_Cmd – для команд более низкого уровня внутри модуля (напр. DO\_Heater1\_Cmd – команда TempManager на 1-й нагреватель, передаваемая ActuatorMgr).
* Переменные с \_Raw (например, PV\_TempProduct\_Raw) – не обработанные величины (но в нашем случае уже переведенные в °C и отфильтрованные от обрыва – возможно, лучше назвать \_Calc или \_Valid). В TЗ решили оставить \_Raw для различия с окончательно фильтрованным PV.
* В именах режимов и состояний используются понятные слова: PasteurActive, CIPAbort, HeaterFault и т.д. Это облегчает чтение кода.

После рефакторинга имен все модули используют единый стиль, противоречий нет. Приложение **Alias-Plan** содержит перечень переименованных сигналов и соответствие старого/нового имени. В частности, HeatingPowerReq переименован в HeatEnable (Pasteur), Mode\_DispCont\_RunReq/Mode\_DispDose\_RunReq укорочены до DispCont\_RunReq в некоторых местах (для удобства), но в коде оставлены как есть ради совместимости с HMI тегами.

Наконец, **запрет прямых действий вне зон ответственности**: Ни один FB не вмешивается в чужие механизмы. Например, DosingManager не пытается управлять клапанами CIP или нагревом – он только ставит насос розлива. PasteurManager не дергает напрямую нагрев – только через TempManager. CIPManager не трогает мешалку напрямую – у него есть сигнал CIP\_Mixer\_ON, а ActuatorManager решает, что с ним делать (в данном случае включает DO3 FWD). Такое чёткое разделение снижает вероятность ошибок и упрощает отладку – каждый блок отвечает только за «свой кусочек» процесса.

Все указанные инварианты будут проверяться в процессе код-ревью и при испытаниях. Любое отклонение (например, обнаружение дублирующей записи переменной, или ситуация, когда при определенном сценарии два режима одновременно активны) трактуется как *ошибка*, и должно быть устранено до ввода в эксплуатацию.