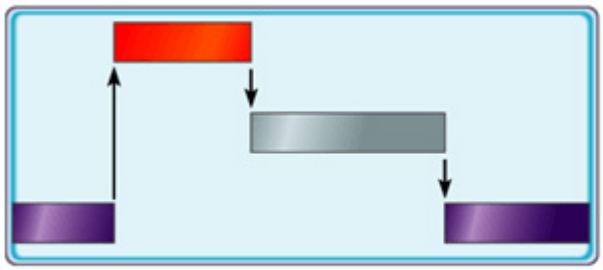
Использование задач в CODESYS V3



27.03.2020
версия 1.0

https://oscat.ru/

Оглавление

Введение

Неотъемлемой частью любого проекта CODESYS является компонент **Конфигурация задач**, с помощью которого производится добавление и настройка задач проекта. Несмотря на важность компонента, его описание в онлайн-справке слишком лаконично, чтобы создать у пользователя полное понимание принципа работы задач и всех касающихся этого процесса особенностей. Отчасти это связано с тем, что реализация управления задачами может отличаться для разных устройств и различных программно-аппаратных платформ. Определенная информация содержится в документе **CODESYS Control V3 Manual**, который предоставляется только производителям оборудования и не предназначен для публичного распространения.

Поэтому целью данного документа является предоставление более полного описания работы с задачами в CODESYS V3.5. Как уже упоминалось, на различных платформах работа с задачами может быть организована разными способами; в рамках данного документа рассматривается платформа **ARM/Linux с PREEMPT RT Patch/CODESYS V3.5**. Примером устройств с такой платформой является современная линейка контроллеров компании ОВЕН.

1. Основная информация о задачах

В стандарте МЭК 61131-3 приводится следующее определение задачи:

«*Задача – это элемент управления, который позволяет выполнять набор POU (программ или экземпляров функциональных блоков, объявленных в программах) на периодической или событийной основе*»

Интересным моментом является упоминание возможности вызова экземпляров функциональных блоков. На практике, в большинстве сред разработки (в том числе, в CODESYS) такая возможность отсутствует. Более того, в стандарте МЭК 61131-8 эта возможность отмечена как *deprecated* (устаревшая).

Задачи помогают пользователю определить, в какие моменты времени будут вызываться его программы. Как правило, в проекте для ПЛК присутствует как минимум одна задача, выполняемая циклически.

В литературе часто приводится понятие цикла (скана) ПЛК, которое сопровождается подобным рисунком:

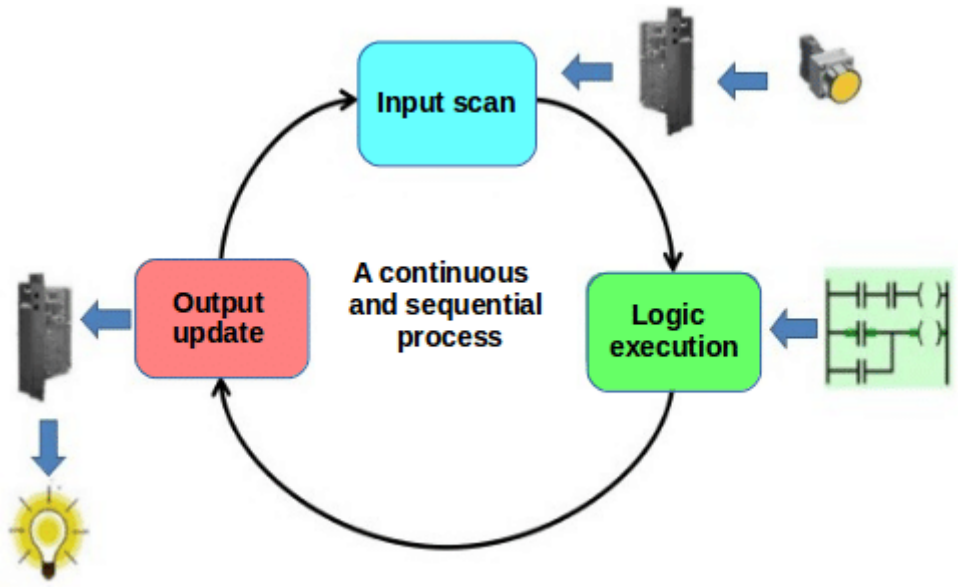


Рис. 1. Цикл работы ПЛК

Подразумевается, что контроллер циклически выполняет три операции:

1. Опрос входов.
2. Выполнение пользовательских программ.
3. Запись значений выходов.

В рамках данной концепции можно считать, что задача определяет период запуска этого цикла.

Подобные рисунки достаточно неплохо описывают поведение типичного однозадачного контроллера, который не имеет сетевых интерфейсов.

Большинство современных контроллеров имеют сетевые интерфейсы и позволяют создавать в рамках проекта несколько задач, что может привести к следующим вопросам:

1. Как производится выполнение нескольких задач по отношению друг к другу?
2. Как по отношению к задачам выполняется сетевой обмен?

Ответы на эти вопросы можно дать только в контексте обзора конкретного ПЛК, выполненного на конкретной платформе.

Главное, что следует отметить – вопрос реализации управления задачами является той областью, в которой соприкасаются особенности реализации системы исполнения (runtime) ПЛК и механизма планирования задач его операционной системы.

2. Реализация управления задачами в CODESYS V3 Runtime

Система исполнения CODESYS V3.5 предоставляет три механизма управления задачами. Разработчик устройства должен выбрать механизм, наиболее подходящий для его программно-аппаратной платформы.

1. *Однозадачность* (single tasking; также этот механизм часто называется *кооперативной многозадачностью*) – все задачи контроллера выполняются последовательно друг за другом в бесконечном цикле. Вызов задач производится планировщиком CODESYS. Порядок выполнения задач определяется их приоритетами. В процессе своего выполнения задача не может быть прервана (то есть она всегда выполняется от начала и до конца). Этот механизм не имеет реализации сторожевого таймера (watchdog).

Данный тип многозадачности обычно используется во встраиваемых устройствах без операционной системы и имеющих крайне ограниченные аппаратные ресурсы. Существенным недостатком данной реализации является тот факт, что коммуникационные функции также выполняются в общем цикле, поэтому проблемы с обменом (например, отсутствие ответа от опрашиваемых устройств) может существенно увеличить джиттер (задержку вызова).

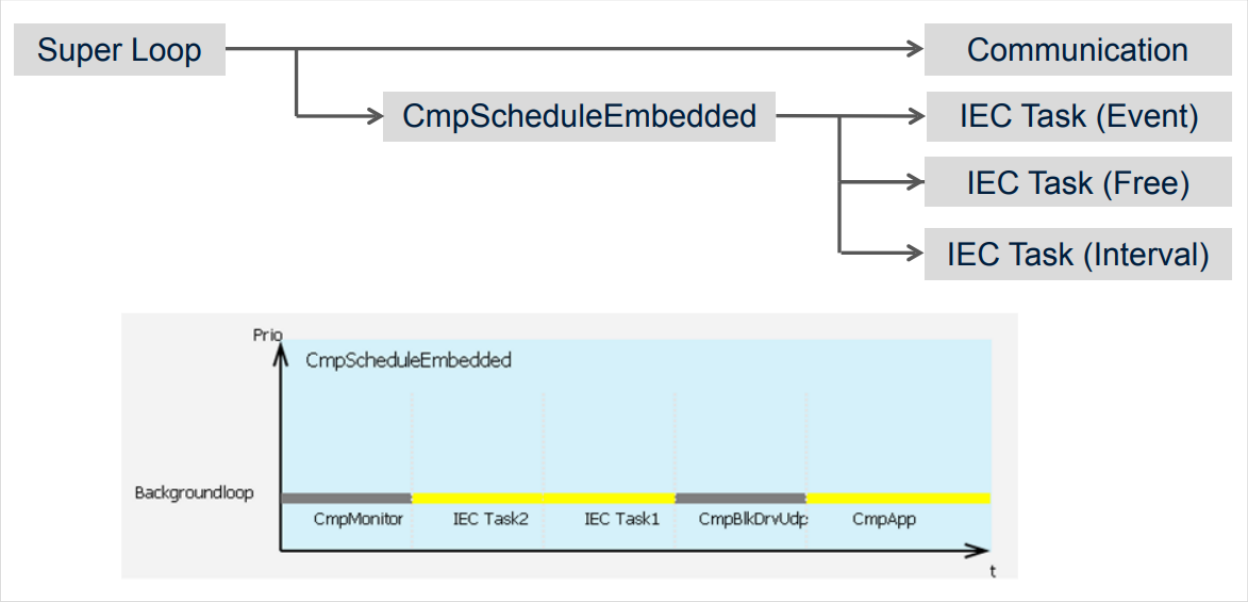


Рис. 2.1. Принцип обработки задач в режиме *Однозадачность*

2. *Планирование по таймеру* (timer scheduler) – вызов каждой задачи происходит периодически с помощью аппаратного таймера, запущенного на устройстве. Некоторые встраиваемые устройства имеют несколько таймеров, которые могут использоваться для этой цели. Поддерживается вытеснение одних задач другими. Данный механизм обычно используется в ПЛК без операционной системы.

Планировщик системы исполнения в данном случае используется только для реализации программного сторожевого таймера.

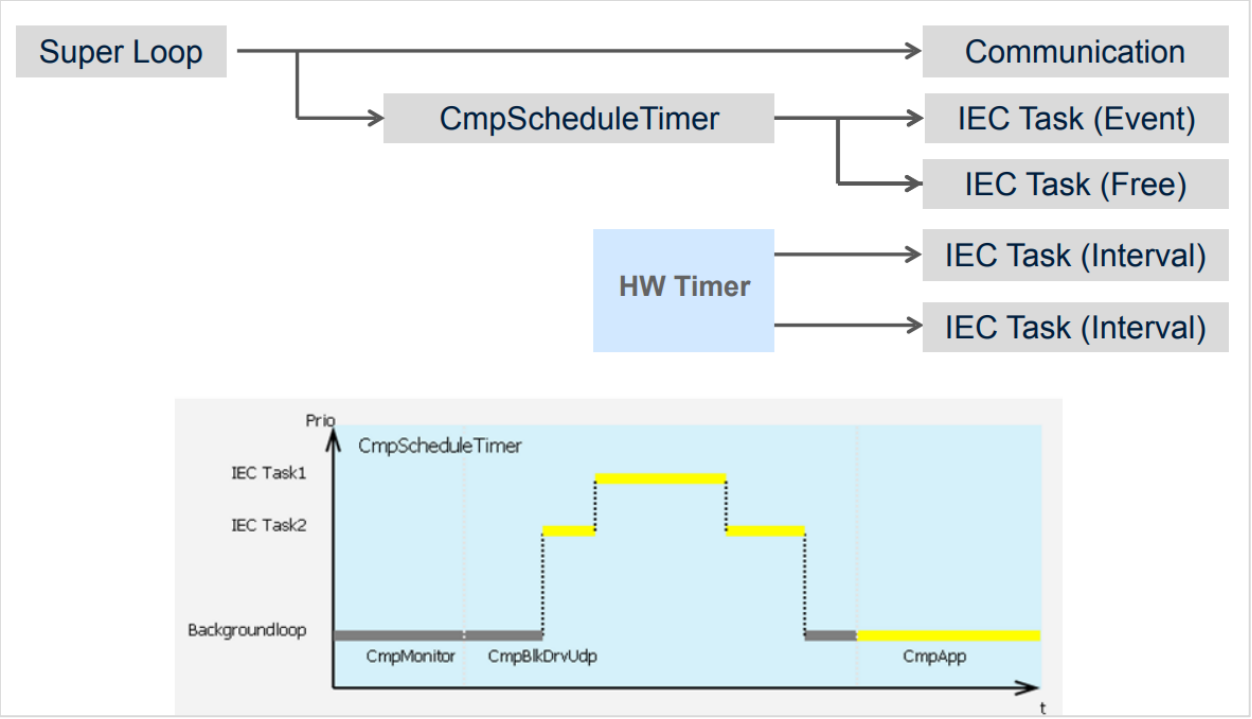


Рис. 2.2. Принцип обработки задач в режиме *Планирование по таймеру*

3. *Вытесняющая многозадачность* (multitasking) – в данном случае каждая задача CODESYS соответствует задаче операционной системы. Диапазон приоритетов задач CODESYS (0...31) соответствует диапазону приоритетов задач реального времени системы исполнения (32...63); подробнее про это см. в приложении А.

В многозадачных системах вызов задач обычно производится планировщиком операционной системы. Но, как правило, такие системы ориентируются на вызов задач по внешним событиям и прерываниям, что не позволяет обеспечить детерминированные интервалы времени между вызовами задач CODESYS. Поскольку контроллеры в большинстве случаев требуют циклический вызов задач с минимальным джиттером, то в системе исполнения CODESYS для их вызова используется отдельный планировщик. Этот же планировщик может выполнять функцию программного сторожевого таймера (так как не каждая ОС имеет встроенный сторожевой таймер).

Если устройство имеет аппаратный сторожевой таймер, то крайне желательно связать его с планировщиком системы исполнения, чтобы детектировать ее зависание.

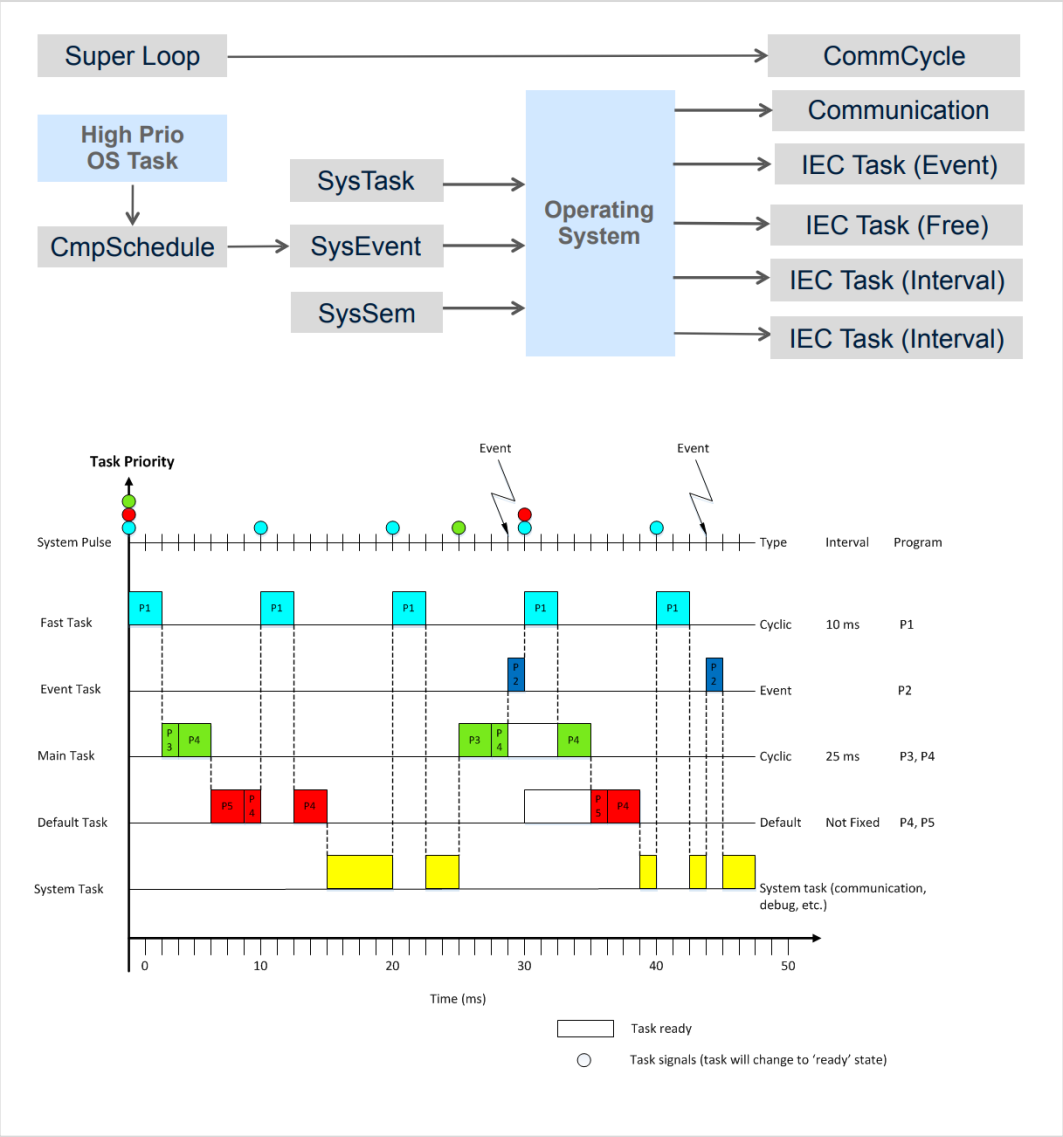


Рис. 2.3. Принцип обработки задач в режиме *Вытесняющая многозадачность* (рис. из [5])

Вызов задач CODESYS в многозадачных системах может выполняться с помощью одного из трех вариантов активации:

3.1. Планировщик CODESYS на каждом тике проверяет наличие задач, для которых подошло время вызова и передает информацию обо всех этих задачах планировщику операционный системы, который производит их вызов в соответствии с приоритетами. Джиттер вызова задач определяется точностью интервалов, с которыми вызывается планировщик CODESYS.

3.2. Аналог варианта 3.1, но происходит вызов только задачи с наивысшим приоритетом. Это упрощает квантование времени (см. ниже), поскольку в каждый момент времени может существовать только одна активная задача.

3.3. Некоторые ОС реального времени (например, Linux с PREEMPT RT Patch) способны вызывать задачи циклически с высокой точностью. В этом случае планировщику системы исполнения достаточно выполнять функции программного сторожевого таймера и не заниматься вызовом задач. Каждая задача CODESYS отображается на свой поток (thread) операционный системы.

Также существует два варианта квантования по времени (timeslicing):

- *Внешнее квантование* – квантование осуществляется внешней (по отношению к системе исполнения CODESYS) задачей. В состоянии ожидания выполнения (suspended) ни одна из задач CODESYS не выполняется. В состоянии возобновления (resume) происходит активация всех задач. Этот механизм базируется на очередях сообщений (SysMsgQ) и, например, может быть применен в ОС VxWorks.
- *Внутреннее квантование* – планировщик системы исполнения CODESYS сам распределяет процессорное время между задачами CODESYS и задачами ОС (например, 80% времени выделяется CODESYS, 20% – задачам ОС).

Примечание для устройств с неточными микросекундными таймерами: планировщик CODESYS допускает, что в системе могут быть ошибки микросекундной синхронизации с отклонением до 25% от заданного периода. Это означает, что для обеспечения детерминированности вызова циклической задачи с интервалом 1 мс необходимо, чтобы рассинхронизация аппаратного таймера и тика планировщика не превышала 25%. Но на некоторых процессорах (например, Cortex A8-ARMs, Via X86 1.2 GHz, Atom с Windows CE 6) эти отклонения могут превышать заданный предел, что приводит к пропуску вызовов задачи (например, 3% вызовов задачи будут пропущены). Для подобных ситуаций в конфиг-файле CODESYS предусмотрена специальная настройка, которая позволяет планировать задачи CODESYS на основе миллисекундных таймеров:

```
[CmpSchedule]
DontUseMicrosecondTiming=1
```

На большинстве контроллеров с ОС Linux (с PREEMPT RT Patch) используется вариант активации **3.3** (планированием задач CODESYS занимается планировщик ОС) и внутреннее квантование. Это справедливо и для контроллеров ОВЕН (ПЛК2xx, СПК1xx).

3. Настройка задач в среде CODESYS V3.5

Настройка задач в CODESYS выполняется через компонент **Конфигурация задач**. Обычно он присутствует в проекте по умолчанию; если компонент отсутствует, то для его добавления следует нажать **ПКМ** на узел **Application** и выбрать команду **Добавление объекта – Конфигурация задач**.

По умолчанию компонент содержит циклическую задачу **MainTask**. Для корректной работы проекта CODESYS в нем должна присутствовать как минимум одна циклическая задача.

Описание компонента Конфигурация задач

Компонент содержит следующие вкладки:

1. Свойства

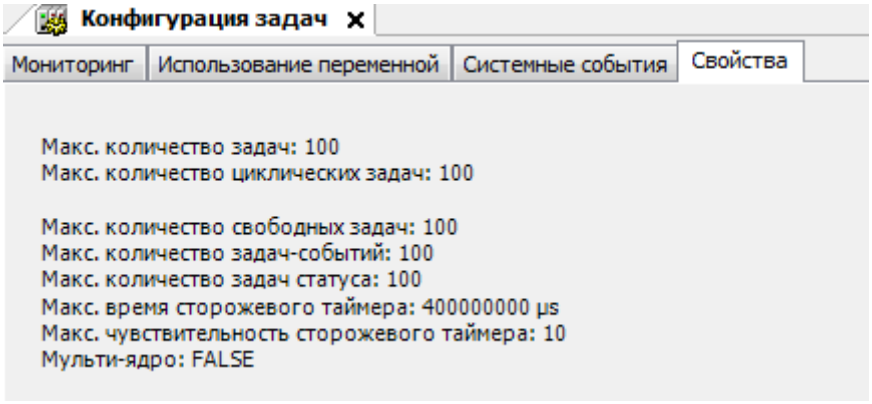


Рис. 3.1. Внешний вид вкладки **Свойства**

На этой вкладке отображается информация о настройках контроллерах, связанных с задачами. Настройки задаются на уровне таргет-файла производителем контроллера (см. приложение Б).

2. Мониторинг

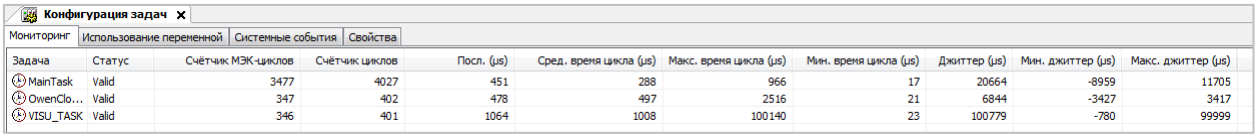


Рис. 3.2. Внешний вид вкладки **Мониторинг**

При подключении к контроллеру на этой вкладке отображается информация онлайн-мониторинга задач – например, время выполнения задачи при ее последнем вызове. Описание параметров вкладки приведено в табл. 3.1.

Для сброса значений следует нажать на строку нужной задачи **ПКМ** и выбрать команду **Сброс**. Следует отметить, что в момент старта приложения все тайминги имеют высокие значения (так как помимо пользовательского кода выполняется код инициализации). Поэтому для реалистичной оценки значений следует сбросить их после начала выполнения приложения.

Для контроллеров ОВЕН информация мониторинга задач также доступна в web-конфигураторе (вкладка **ПЛК/Приложение**).

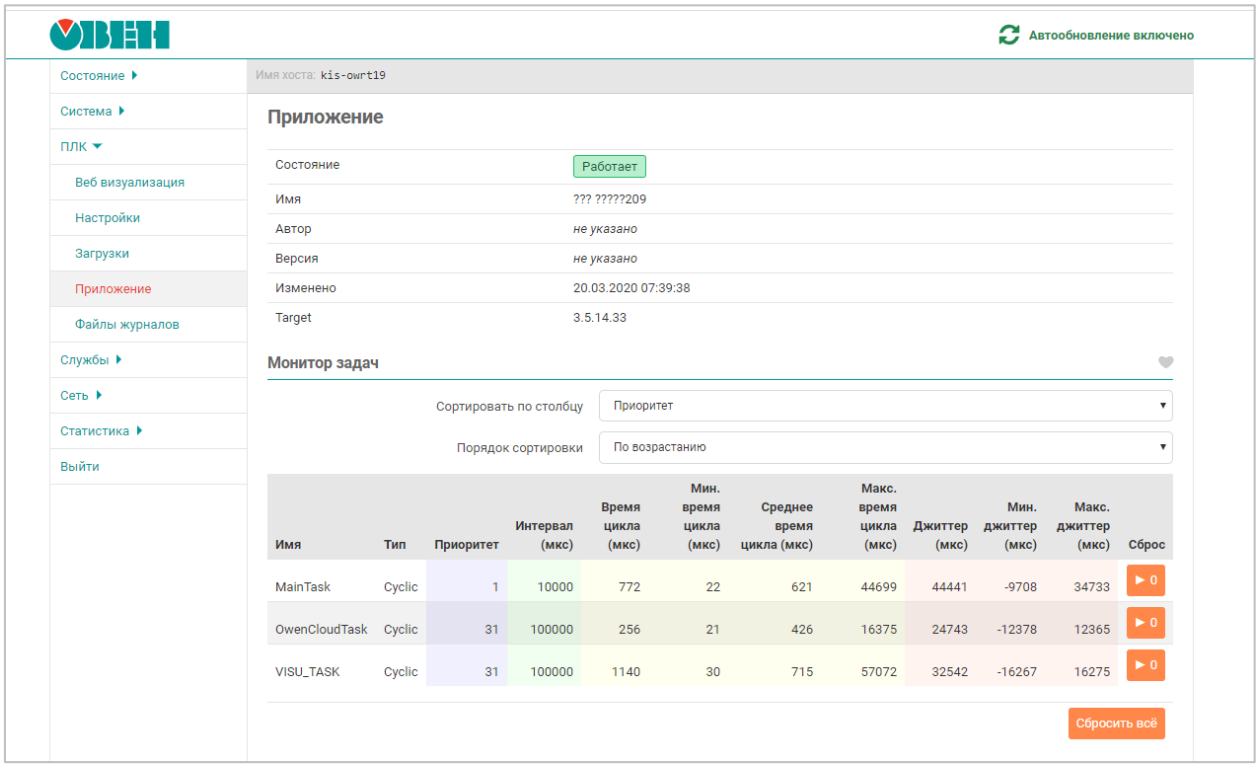


Рис. 3.3. Отображение информации мониторинга задач в web-конфигураторе для ПЛК ОВЕН

3. Использование переменной

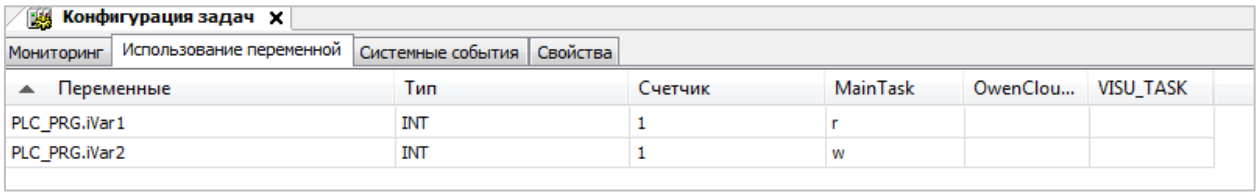


Рис. 3.4. Внешний вид вкладки **Использование переменной**

На этой вкладке выводится информация по использованию переменных в задачах проекта. Для каждой переменной отображаются задачи, в которых она используется (параметр **Счетчик** показывает количество задач, в которых используется данная переменная) и тип доступа (**r** – чтение, **w** – запись).

Информация на вкладке обновляется после каждой компиляции проекта.

4. Системные события

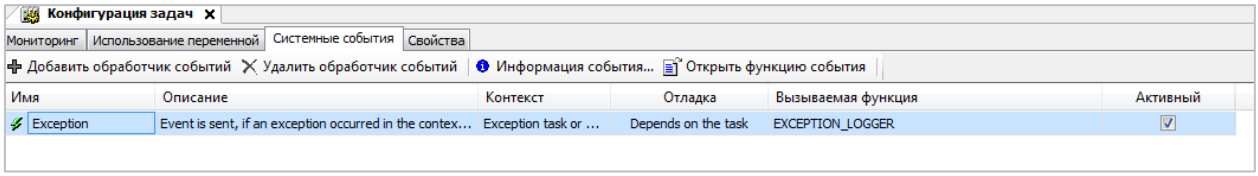


Рис. 3.5. Внешний вид вкладки **Системные события**

На этой вкладке выполняется настройка обработки системных событий. Системным событием, например, является возникновение исключение, загрузка проекта, подключение к контроллеру и т.д. Для системного события можно создать функцию-обработчик, которая будет вызвана при его возникновении. Обратите внимание, что функция-обработчик создается при нажатии на кнопку **Добавить обработчик событий**, т.е. не следует пытаться создать ее заранее.

Для контроллеров с многоядерной (multicore) системой исполнения CODESYS в компоненте **Конфигурация задач** доступны две дополнительные вкладки:

5. Группы задач

На этой вкладке происходит распределение задач между ядрами CPU.

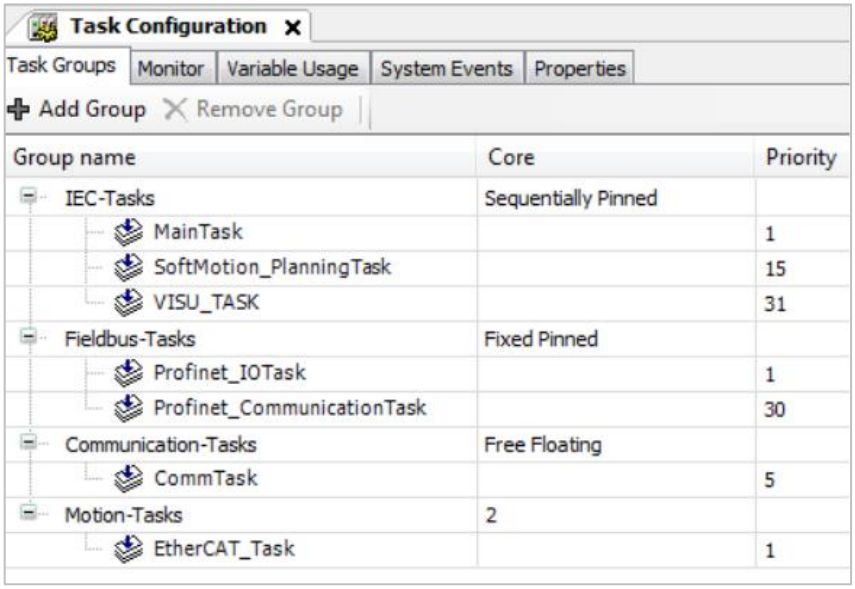


Рис. 3.6. Внешний вид вкладки **Группы задач**

6. Загрузка CPU

На этой вкладке отображаются графики загрузки ядер процессора.

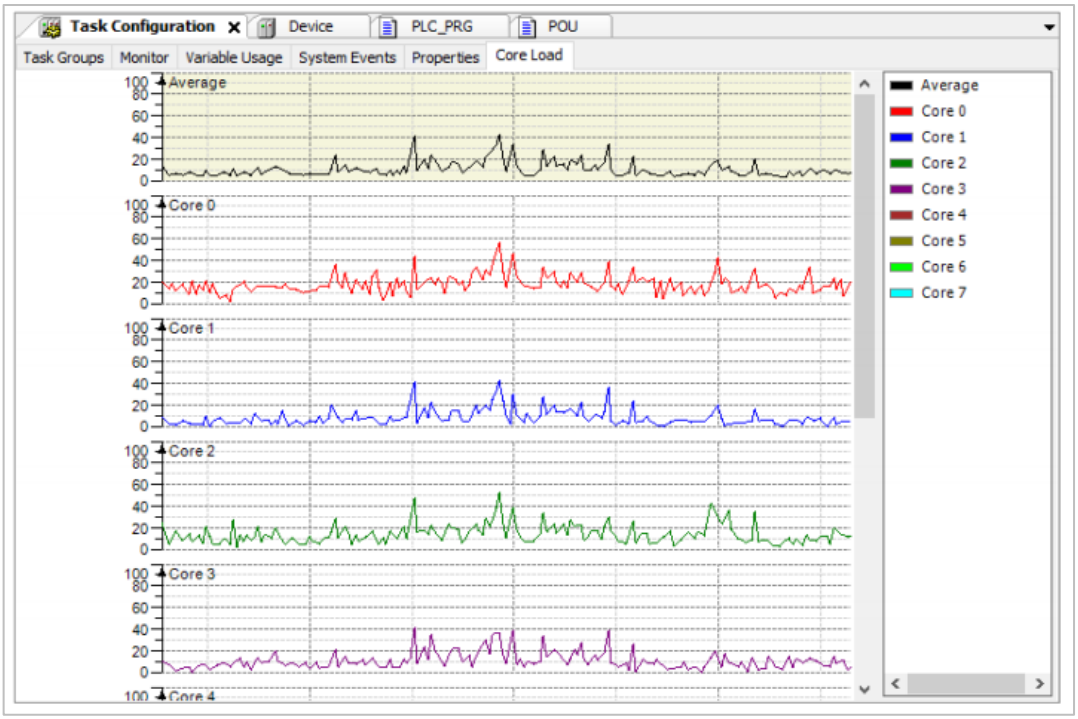


Рис. 3.7. Внешний вид вкладки **Загрузка CPU**

Описание настроек задачи

Некоторые задачи добавляются в проект автоматически при добавлении в проект определенных компонентов (например, трендов, конфигурации тревог, EtherCAT Master и т.д.). Также пользователь может добавлять задачи вручную.

Для создания новой задачи следует нажать **ПКМ** на узел **Конфигурация задач** и выбрать команду **Добавление объекта – Задача**.

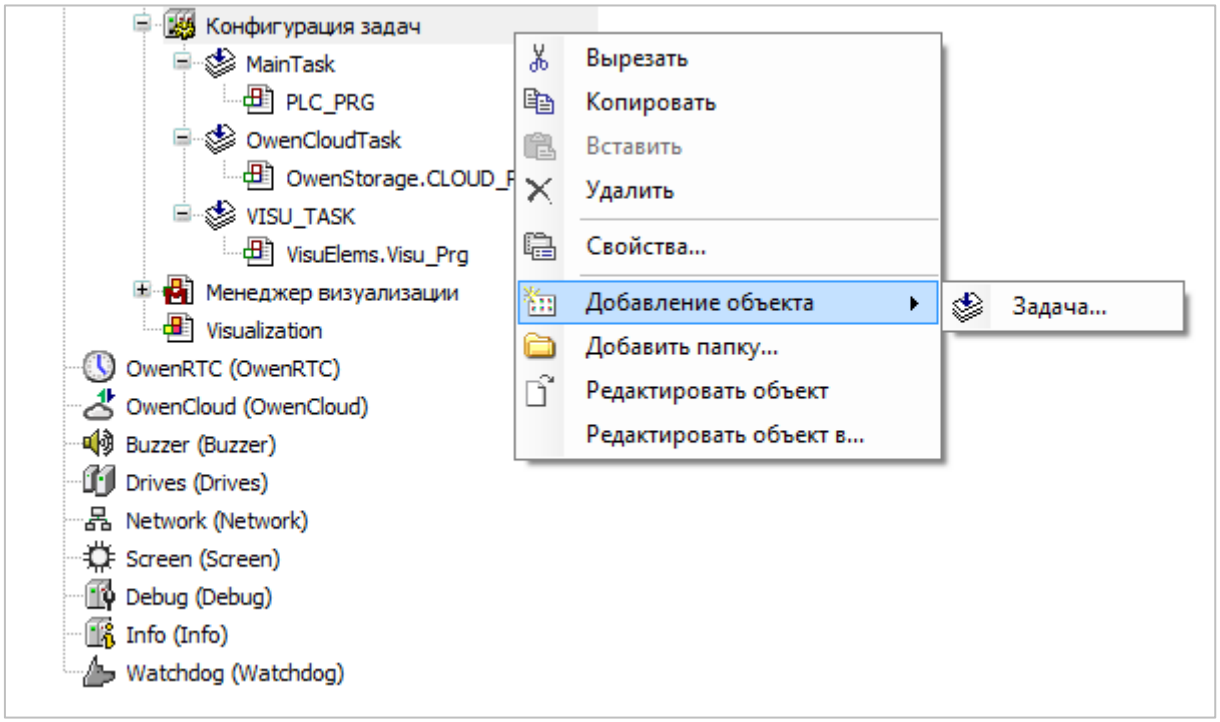


Рис. 3.8. Добавление задачи в проект CODESYS

В верхней части окна задачи происходит работа с ее настройками. В нижней части окна происходит добавление программ, которые будут вызываться в данной задаче. К задаче может быть привязано неограниченное количество программ.

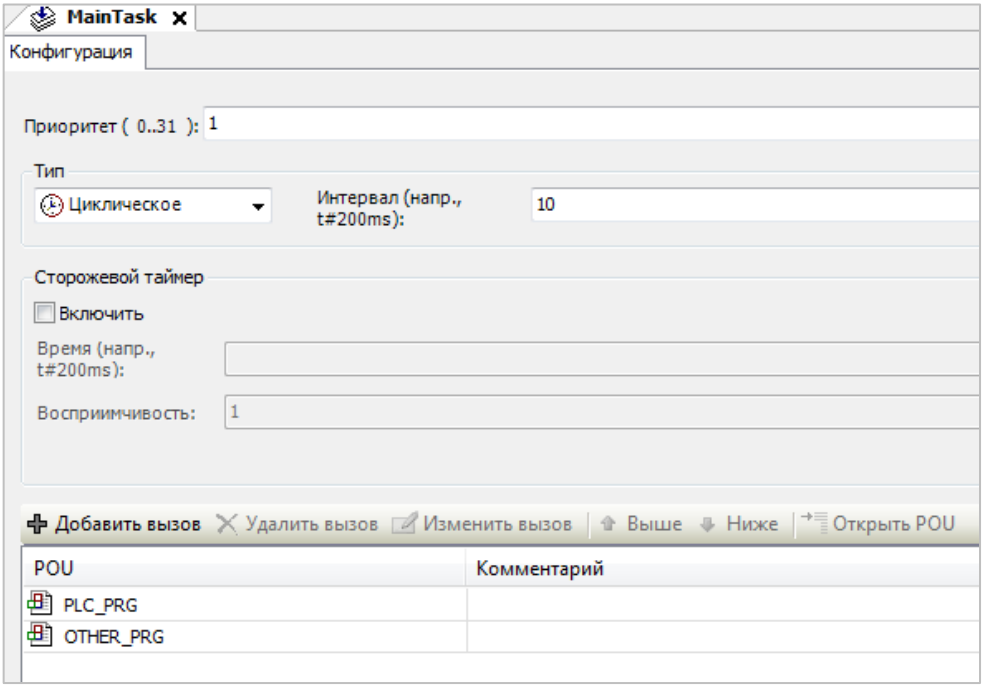


Рис. 3.9. Настройки задачи

Задача имеет следующие настройки:

1. *Приоритет* – значение, которое характеризует важность данной задачи. По умолчанию (*если это не изменено производителем устройства*) выбирается из диапазона 0…31, где **0** – наибольший приоритет, **31** – наименьший. Роль приоритета заключается в следующем: если происходит ситуация, при которой в один и тот же момент времени выполнились условия вызова нескольких задач, то вызвана будет только задача с наибольшим приоритетом.

Для контроллеров с ОС Linux с PREEMPT RT Patch задачи с приоритетом 0…15 выполняются с классом планирования **SHED_FIFO** (задачи реального времени), а задачи с приоритетом 16…31 – с классом планирования **SHED_OTHER**. Более подробную информацию о классах планирования и работе планировщика Linux можно найти в сети (например, см. эту статью).

2. *Тип* – тип вызова задачи. Возможные варианты:

- *Циклическое* – задача вызывается циклически через заданные интервалы времени. Фактический период вызова может отличаться от заданного пользователем – например, ресурсов контроллера может не хватать для выполнения всех задач за заданные интервалы. Для определения реального времени выполнения задачи и задержек ее вызова следует использовать вкладку **Мониторинг** компонента **Конфигурация задач** (см. рис. 3.2).
- *Событие* – задача однократно вызывается по переднему фронту заданной пользователем переменной типа **BOOL**. Переменная должна изменяться в контексте другой задачи.

- *Свободное выполнение* – задача выполняется циклически без задаваемого интервала: сразу после завершения выполнения задачи происходит ее повторный вызов. Строго говоря, приведенная формулировка является упрощенной: при такой реализации задачи с более низким приоритетом (по сравнению с задачей свободного выполнения) никогда бы не выполнялись. Чтобы избежать этого, обработка задач свободного выполнения происходит по одному из следующих вариантов:
  1. Вариант по умолчанию – после выполнения задача предоставляет определенное время для выполнения других задач (это время составляет 20% от времени выполнения задачи).
  2. Если в конфиг-файле CODESYS определена максимальная загрузка процессора, то предоставляемое другим задачам время вычисляется по формуле:
  (100% – максимальная загрузка процессора) · время выполнения задачи

```
[CmpSchedule]
ProcessorLoad.Maximum=80
```

  3. Время, предоставляемое другим задачам, может быть задано в конфиг-файле CODESYS в явном виде (в мс).

```
[CmpSchedule]
Task.Freewheeling.Cycletime=10
```

- *Статус* – задача вызывается в режиме свободного выполнения, пока выбранная пользователем переменная типа **BOOL** имеет значение **TRUE**. Переменная должна изменяться в контексте другой задачи. Если переменная принимает значение **FALSE** – то задача перестает вызываться.
- *Внешнее событие* – задача однократно вызывается по внешнему событию ОС (например, по прерыванию). Этот тип выполнения доступен только в том случае, если он поддержан производителем контроллера.

3. *Сторожевой таймер* (watchdog) позволяет детектировать «зависание» задачи. В стандартной реализации CODESYS в этом случае выполняется генерация исключения и перевод контроллера в состояние СТОП (в этом случае перестают выполняться все задачи контроллера, а не только зависшая). Производитель контроллера может реализовать свою обработку исключений – например, для контроллеров ОВЕН можно настроить перезагрузку ПЛК при возникновении подобных ситуаций.

Сторожевой таймер имеет два параметра – время (обозначим его как T) и восприимчивость (обозначим его как N) . Генерация исключения о зависании задачи будет произведена в двух случаях:

- если в течение N последовательных циклов вызова задачи ее фактическое время выполнения будет превышать T;
- если в течение одного вызова задачи ее время выполнения превысит N·T.

Описание обработки задач для контроллеров с Linux с PREEMPT RT Patch

Планировщик задач (им может быть планировщик CODESYS или планировщик ОС, п. 2) в каждом своем вызове проверяет условия вызова задач CODEYS. Если условие выполняется – то происходит вызов задачи. Если одновременно выполняются условия нескольких задач – то вызывается задача с наивысшим приоритетом. Если в момент выполнения условия уже выполняется какая-либо задача – то происходит сравнение приоритетов:

- если выполняемая в данный момент задача имеет более высокий приоритет, то задача с выполнившимся условием добавляется в список ожидания;
- если выполняемая в данный момент задача имеет более низкий приоритет, то она приостанавливается и начинается выполнение задачи с более высоким приоритетом. После выполнения высокоприоритетной задачи низкоприоритетная задача будет возобновлена (если в этот момент не активируется условие выполнения другой высокоприоритетной задачи).

Таким образом, задача CODESYS не обязательно выполняется «от начала и до конца» - ее выполнение может прерываться другими более высокоприоритетными задачами (выполнение которых также может прерываться). В этом и заключается механизм вытесняющей многозадачности.

Если происходит одновременное выполнение условия задач с одинаковым приоритетом, то из них вызывается та, которая к данному моменту дольше всех ожидает своего вызова.

Программы, привязанные к задаче, вызываются последовательно в порядке своего добавления (см. рис. 3.9).

Табл. 3.1. Параметры вкладки **Мониторинг** компонента **Конфигурация задач**

Параметр	Описание
Статус	Статус задачи. Возможные значения: **Not created** – задача еще ни разу не выполнялась (характерно для событийных задач); **Generated** – задача зарегистрирована в системе исполнения, но еще ни разу не вызывалась; **Valid** – задача выполняется в нормальном режиме; **Exception** – в контексте задачи произошло исключение, выполнение задачи было приостановлено.
Счетчик МЭК-циклов	Количество произошедших вызовов задачи (с момента запуска приложения), в которых происходил вызов POU задачи
Счетчик циклов	Общее количество произошедших вызовов задачи с момента запуска приложения (включая те, в которых не происходил вызов POU). В зависимости от конкретной реализации для конкретного устройства – значения счетчика циклов и счетчика МЭК-циклов могут как совпадать, так и отличаться.
Посл.	Фактические время выполнения задачи при ее последнем вызове (в микросекундах)
Сред. время цикла	Усредненное по всем произошедшим МЭК-циклам время выполнения задачи (в микросекундах)
Макс. время цикла	Максимальное время выполнения задачи (в микросекундах)
Мин. время цикла	Минимальное время выполнения задачи (в микросекундах)
Джиттер	Джиттер (см. ниже) задачи для ее последнего вызова (в микросекундах)
Мин. джиттер	Максимальное (по модулю) время опережения (см. ниже) вызова задачи (в микросекундах)
Макс. джиттер	Максимальное время задержки вызова задачи (в микросекундах)

Как уже упоминалось, к моменту выполнения условия вызова низкоприоритетной задачи может все еще выполняться высокоприоритетная задача, поэтому вызов низкоприоритетной задачи будет отложен. Задержка между моментом выполнения условия вызова задачи и ее фактическим вызовом называется **джиттером**. Планировщик задач может стараться компенсировать джиттер, смещая следующий вызов задачи на более ранний момент времени (т.е. вызывая ее с опережением).

Обратите внимание, что в настройках задачи интервал ее вызова обычно задается в **милли**секундах, а на вкладке мониторинга значения таймингов отображаются в **микро**секундах.

4. Задачи и коммуникационные драйверы

Пользователь CODESYS может настроить обмен с другими устройствами с помощью компонентов, добавляемых через дерево проекта (**ПКМ** на узел **Device** – **Добавить устройство**). Добавление некоторых компонентов (например, компонентов протоколов EtherCAT и Profinet) приводит к автоматическому созданию задач, в цикле которых выполняется обмен по данному протоколу.

Но для некоторых компонентов (например, компонентов протокола Modbus) никаких отдельных задач не создается. Соответственно, возникает вопрос – в цикле каких задач вызываются эти компоненты?

Рассмотрим это на примере проекта, в котором контроллер работает в режиме **Modbus TCP Master**. В дерево проекта добавлены следующие компоненты:

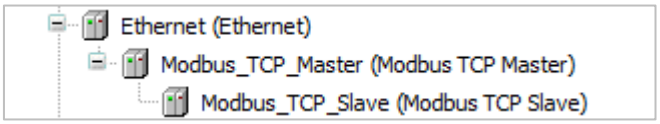


Рис. 4.1. Настройка контроллера в режиме **Modbus TCP Master**

В компоненте **Modbus TCP Master** на вкладке **Modbus TCP Master Соотнесение входов/выходов** можно выбрать конкретную задачу, в цикле которой будет вызываться данный компонент.

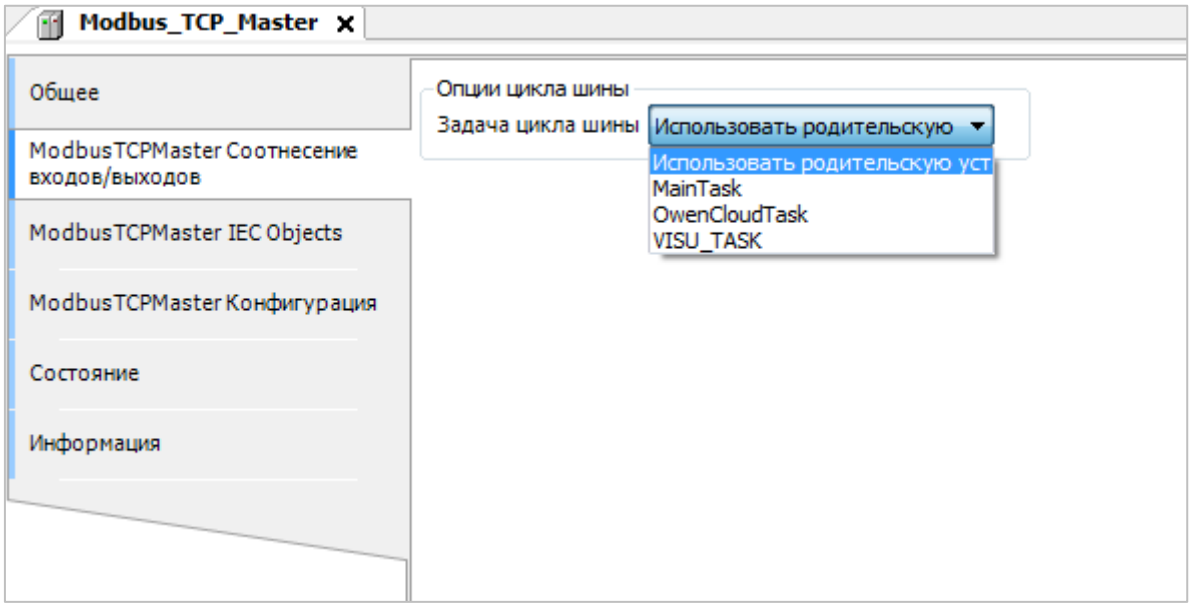


Рис. 4.2. Выбор задачи цикла шины для компонента **Modbus TCP Master**

По умолчанию установлен вариант **Использовать родительскую установку**. Это значит, что данный компонент будет вызываться в задаче своего родительского компонента. Для компонента **Modbus TCP Master** таковым является компонент **Ethernet** (см. рис. 4.1).

У компонента **Ethernet** есть точно такая же настройка:

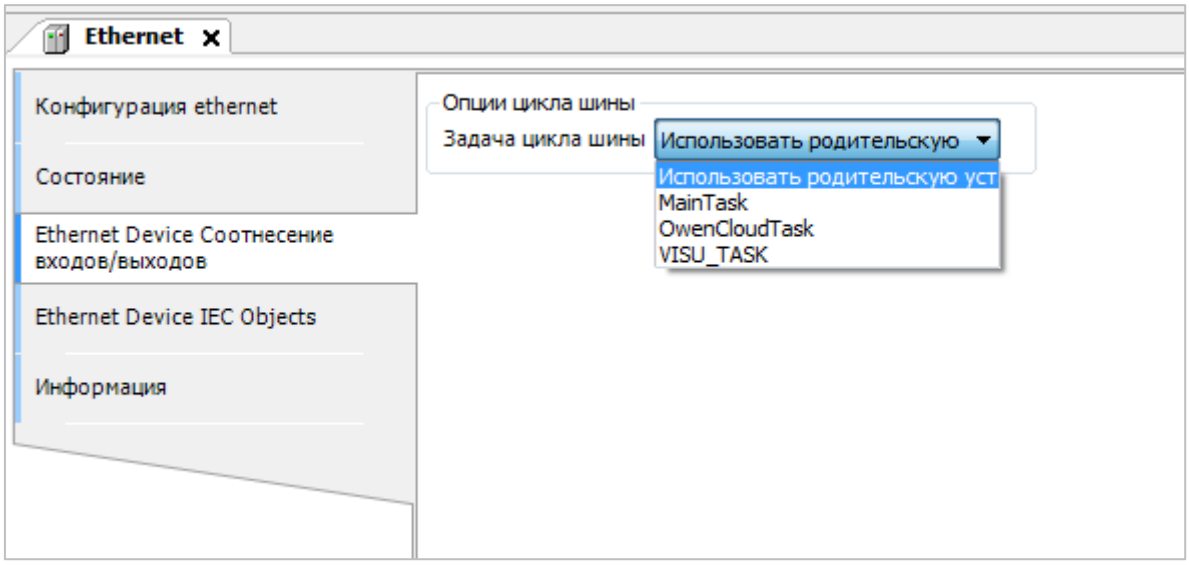


Рис. 4.3. Выбор задачи цикла шины для компонента **Ethernet**

Как мы видим, в этом компоненте тоже по умолчанию тоже используется вариант **Использовать родительскую установку**. Но родительского компонента у компонента **Ethernet** нет. Что же произойдет в этом случае?

В такой ситуации родительским компонентом будет считаться узел Device, у которого на вкладке **Установки ПЛК** можно выбрать задачу цикла шины:

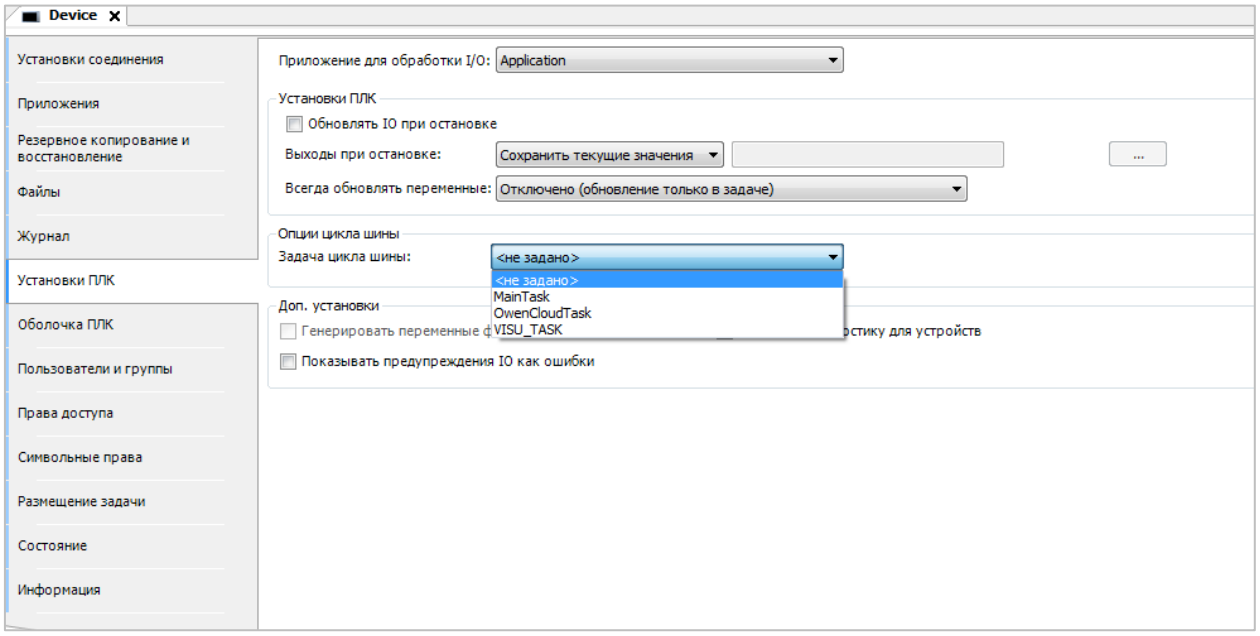


Рис. 4.4. Выбор задачи цикла шины в узле **Device**

По умолчанию данный параметр имеет значение **<не задано>.** Это означает, что в качестве задачи цикла шины используется задача проекта с наименьшим временем цикла (обычно такой задачей является задача **MainTask**).

Таким образом, если в проекте есть задача с адекватным временем цикла (например, для протокола Modbus – 10…20 мс) – то описанные выше настройки задач всех Modbus-компонентов можно оставить в значениях по умолчанию, и при этом никаких проблем с обменом не возникнет.

С другой стороны, если пользователь, например, увеличит время задачи MainTask до 100 мс (и при этом в проекте не будет задач с меньшим временем цикла) – то обмен будет работать некорректно (с точки зрения пользователя опрос будет медленным, часть ответов slave-устройств будет пропущена).

Всё описанное выше справедливо и для компонентов **Modbus Serial** (Modbus COM, Modbus Serial Com Port) – с тем исключением, что у компонента **Modbus COM** отсутствует вкладка **Соотнесение входов/выходов** (так как COM-порт однократно открывается при запуске CODESYS и не требует циклического вызова).

В баг-трекере CODESYS уже несколько лет зафиксировано пожелание по автоматическому созданию коммуникационной задачи для Modbus-компонентов, но пока оно не запланировано к реализации.

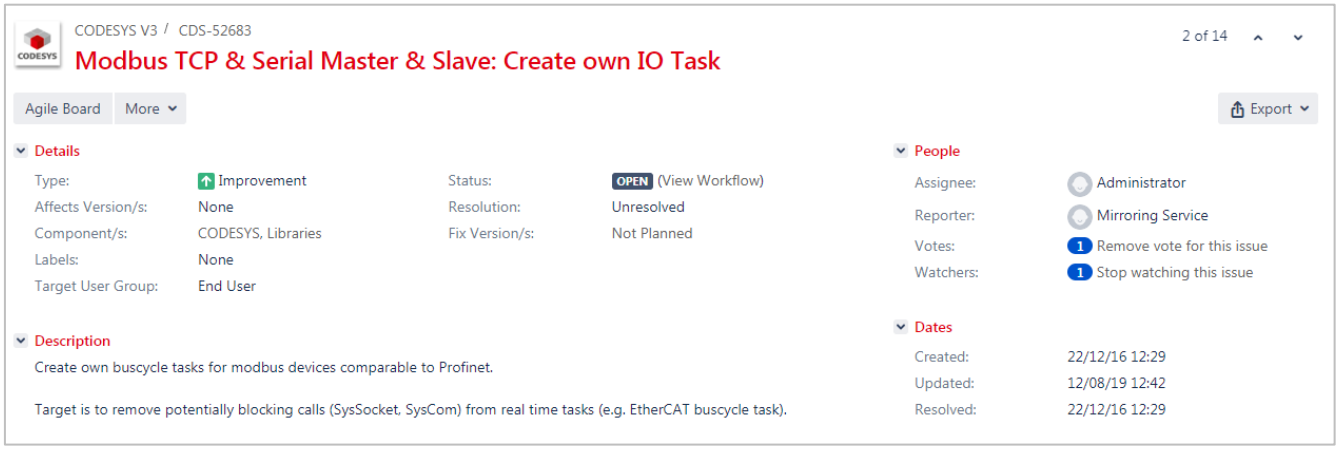


Рис. 4.5. Пожелание по автоматическому созданию задачи Modbus в баг-трекере CODESYS

Отметим еще один момент – у компонентов **Modbus Slave Com Port, Modbus TCP Slave**, **Modbus Serial Device** и **Modbus TCP Device** на вкладе **Соотнесение входов-выходов** присутствует параметр Всегда обновлять переменные.

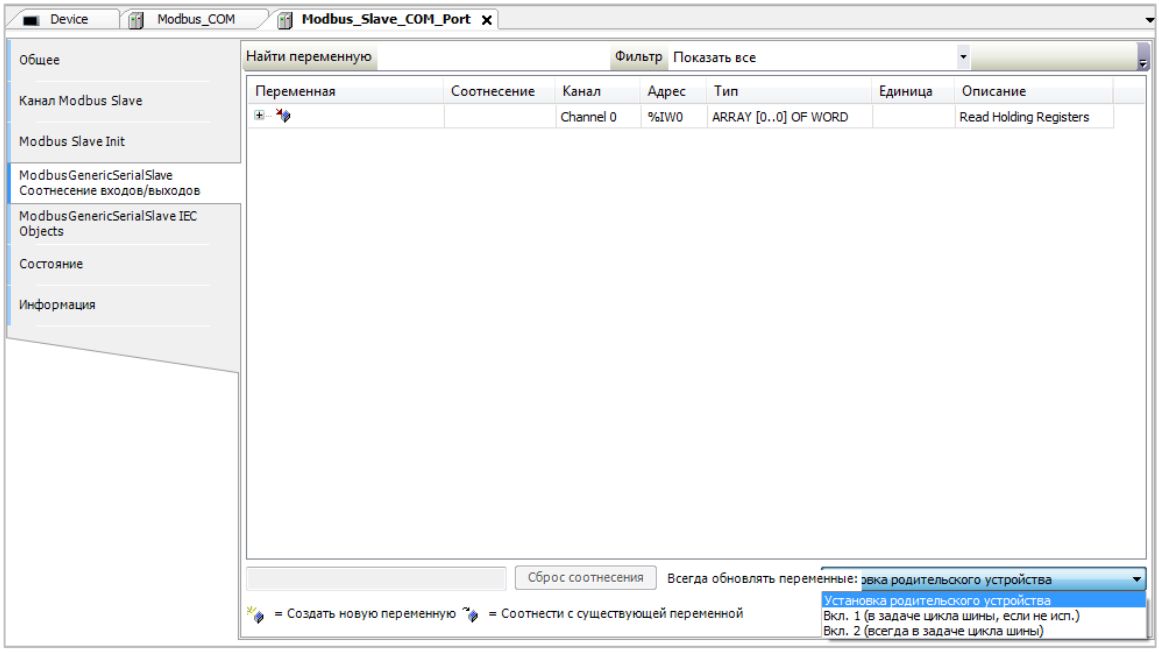


Рис. 4.6. Настройка **Всегда обновлять переменные**

Этот параметр определяет, когда происходит обновление данных в каналах компонента, к которым привязываются переменные проекта. Возможные значения:

- *Установка родительского устройства* – для каналов с привязанными переменными обновление происходит в задаче родительского компонента, для каналов с непривязанными переменными – обновления не происходит;
- *Вкл. 1 (в задаче цикла шины, если не исп.)* – если переменная не используется в коде программы, связанной с какой-либо задачей, то обновление происходит в задаче цикла шины. Если используется – обновление происходит в цикле задачи, в программе которой используется переменная. Обновление каналов с непривязанными переменными происходит в задаче цикла шины;
- *Вкл. 2 (всегда в задаче цикла шины)* – обновление всех каналов происходит в задаче цикла шины.

При отладке обмена, когда переменные просто объявлены в программах, но еще не написан никакой код (или к каналам компонентов вообще еще ничего не привязано), у пользователей часто возникает проблема – несмотря на отсутствие каких-то проблем с обменом все каналы имеют значение «0» и выделяются серым цветом (даже если фактически в них записываются какие-то значения).

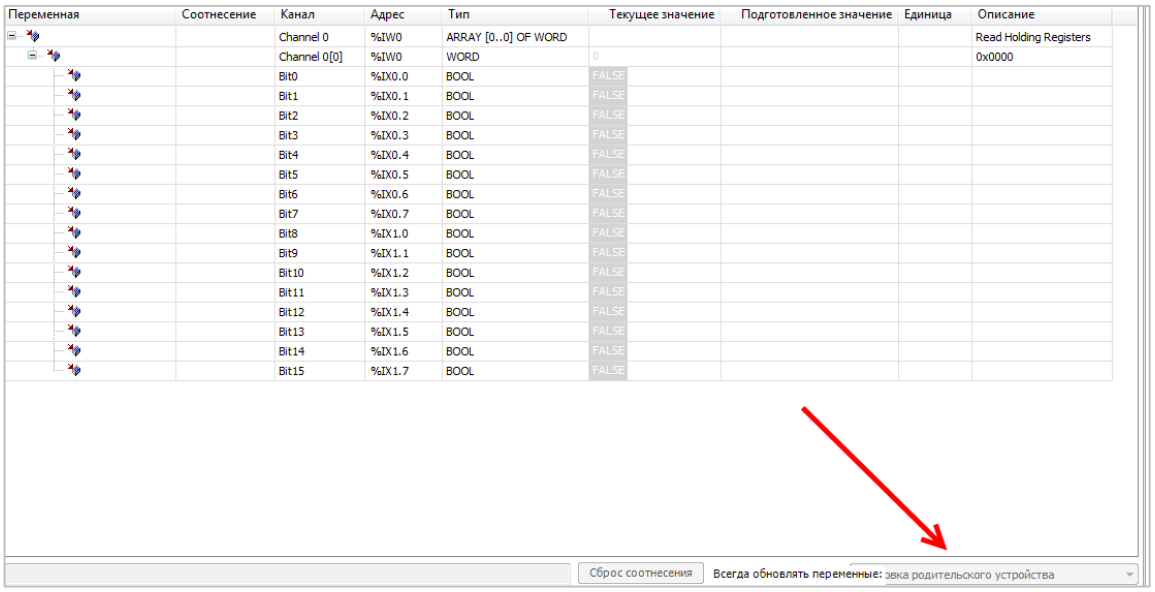


Рис. 4.7. Внешний вид необновляемых каналов

Для решения этой проблемы необходимо для параметра **Всегда обновлять переменные** установить значение **Вкл. 2 (Всегда в задаче цикла шины)** - см. рис. 4.6.

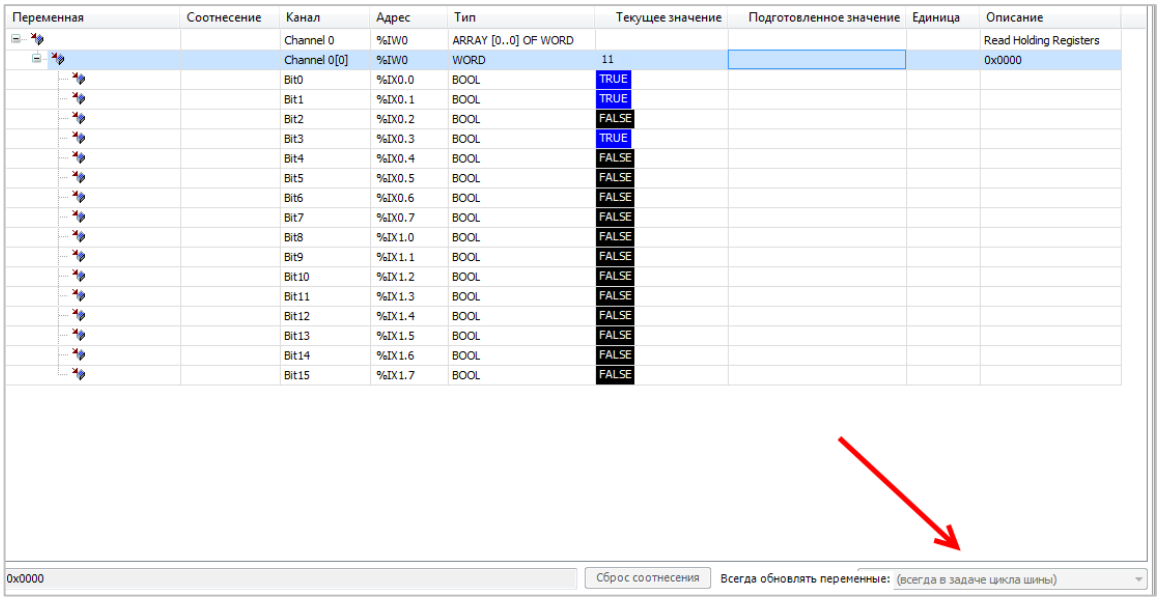


Рис. 4.8. Внешний вид обновляемых каналов

У внимательных пользователей может возникнуть вопрос – а как именно компоненты Modbus обрабатываются в контексте вызывающей их задачи? Точное описание работы коммуникационных драйверов отсутствует, но в целом можно считать, что операции, связанные с обменом (например, чтение/запись из COM-порта) выполняются асинхронно по отношению к задаче цикла шины. В задаче CODESYS происходит только синхронизация буферов драйверов с каналами компонентов. В справке CODESYS приводится следующий рисунок, который поясняет принцип работы драйверов:

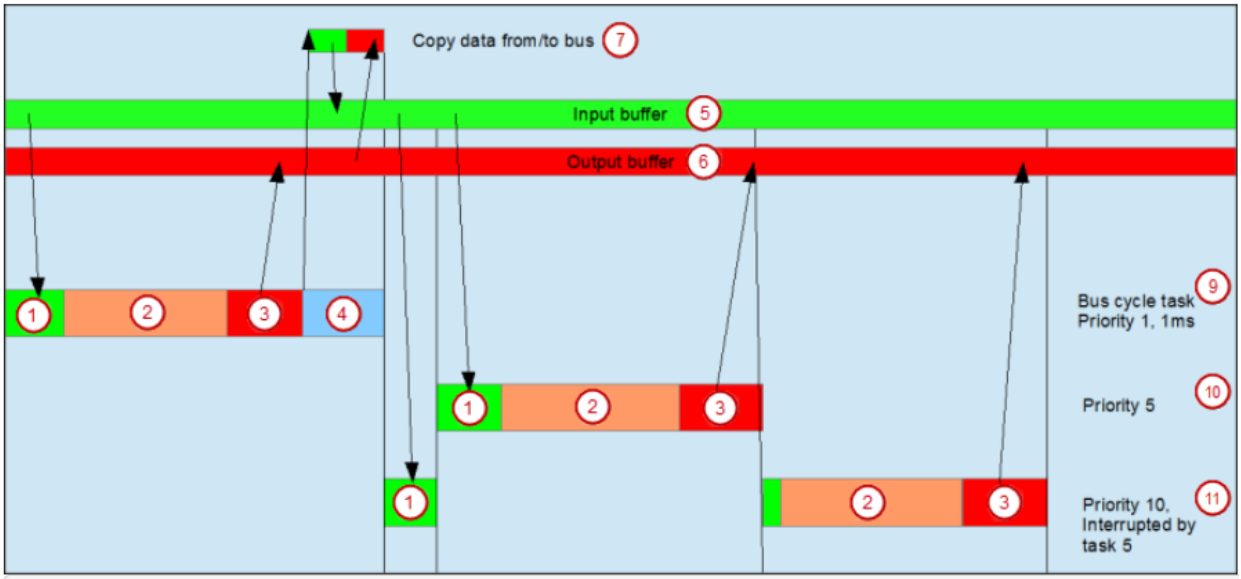


Рис. 4.9. Принцип работы коммуникационных драйверов

5. Многоядерная система исполнения CODESYS Multicore

Начиная с версии CODESYS V3.5 SP13 в CODESYS появилась поддержка многоядерных систем исполнения (*строго говоря, поддержка бета-версии технологии была уже в SP12*).

Для многоядерных систем исполнения в компоненте **Конфигурация задач** доступны дополнительные вкладки, позволяющие распределять задачи по конкретным ядрам процессора (см. п. 3, рис. 3.6).

Больше информации по этому вопросу приведено в обзоре на сайте CODESYS и онлайн-справке CODESYS (общее описание, конкретные аспекты).

6. Синхронизация данных между задачами

Мы уже упоминали, что один из вариантов реализации многозадачности в CODESYS – это вытесняющая многозадачность, при которой высокоприоритетная задача может приостанавливать выполнение низкоприоритетной. При этом в какой-то момент времени выполнение низкоприоритетной задачи будет возобновлено. Если задачи работают с одними и теми же данными – то это может привести к ряду проблем, например:

- вытесненная задача не успела до конца сформировать сообщение в буфере данных, а вытеснившая ее задача пытается работать с этим буфером;
- вытеснившая задача произвела изменение части данных вытесненной задачи, что может привести к некорректным вычислениям после возобновления работы вытесненной задачи (например, обрабатываемые данные изменились, а их коэффициенты – нет).

По возможности, подобных ситуаций (когда разные задачи работают с одними и теми же данными) лучше избегать. Если обойтись без этого затруднительно – то следует организовать синхронизацию данных между задачами.

В CODESYS V3.5 SP15 появился специальный компонент для этих целей – Список глобальных переменных (task-local). Также информация по синхронизации данных между задачами приведена в этом разделе онлайн-справки.

7. Работа с задачами из кода программы

Для работы с задачи из кода программы используется библиотека **CmpIecTask**. Библиотека позволяет создавать задачи, удалять их, включать/отключать сторожевой таймер, считывать информацию мониторинга задач в переменные программы. См. видеопример использования библиотеки.

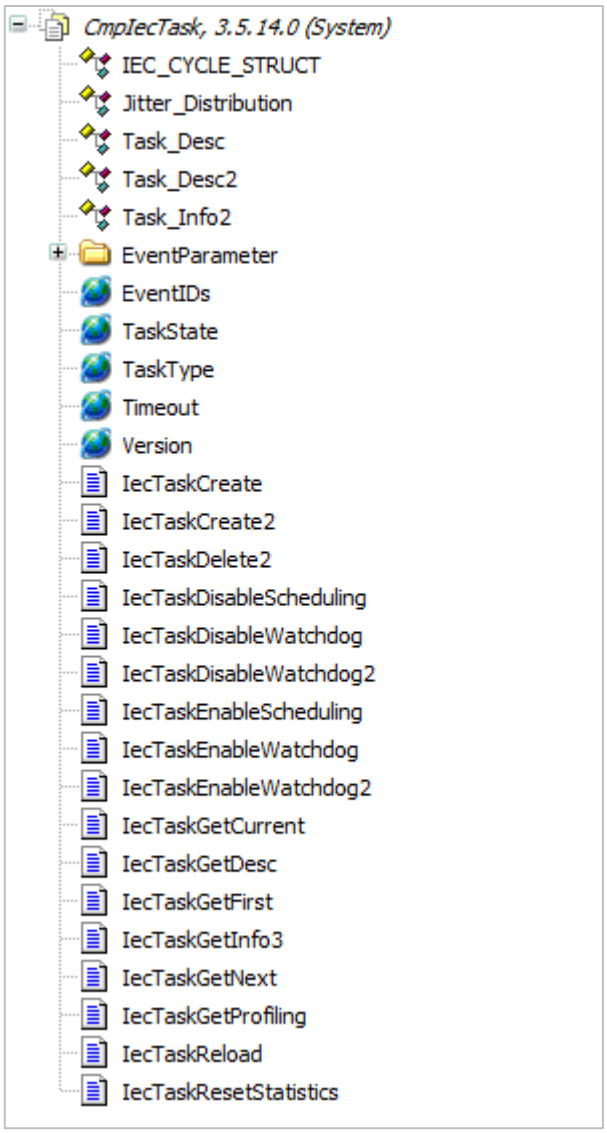


Рис. 7.1. Состав библиотеки **CmpIecTask**

8. Рекомендации по работе с задачами

1. Не добавляйте в проект задачи (используйте только задачи, автоматически создаваемые CODESYS).
2. Если вы добавляете в проект задачу – то должны четко понимать, как именно реализована обработка многозадачности в CODESYS для используемого вами ПЛК и уметь ясно ответить на вопрос, зачем именно вы создаете эту задачу.

Опишем две типичные ситуации, которые не требуют создания новых задач:

- В проекте есть задача **MainTask** с интервалом 10 мс, но некоторые операции требуется выполнять значительно реже (например, раз в 1000 мс). Вместо создания новой циклической задачи можно добавить в задачу **MainTask** генератор импульсов, который будет срабатывать раз в 1000 мс, и использовать выход этого генератора в качестве условия выполнения нужных операций.
- В проекте есть задача **MainTask** с интервалом 10 мс, но некоторые операции требуется выполнять только при возникновении определенного события (по переднему фронту переменной, которая соответствует этому событию). Вместо создания задачи типа **Событие** можно добавить в задачу **MainTask** необходимую логику, использовав для детектирования импульсов переднего фронта ФБ **R_TRIG** из библиотеки **Standard**.

3. Не назначайте задачам тип **Свободное выполнение** или **Статус**.
4. Не изменяйте интервалы вызова и приоритеты задач, которые CODESYS добавляет автоматически.
5. В проекте должна присутствовать хотя бы одна задача с адекватным в рамках вашей системы управления интервалом вызова (обычно такой задачей является **MainTask**, которая вызывается с интервалом не менее 20 мс).
6. Не редактируйте в компонентах Modbus задачу цикла шины.
7. Не добавляйте в одной задаче несколько вызовов одной и той же программы.
8. Не добавляйте в разных задачах вызовы одной и той же программы.
9. Если программа вызывается задачей – то она не должна вызываться другими программами.
10. По возможности избегайте передачи данных между задачами. Если избежать этого не удается – организуйте синхронизацию данных между задачами (см. ссылки в п. 6).

Приложение А. Диапазон приоритетов системы исполнения CODESYS

Приоритеты задач системы исполнения CODESYS разбиты на 8 сегментов:

Название сегмента	Диапазон приоритетов сегмента	Описание
TASKPRIO_SYSTEM	0…31	Системные задачи (например, задача планировщика)
TASKPRIO_REALTIME	32…63	Задачи проекта CODESYS. Приоритеты 0…31, задаваемые в компоненте **Конфигурация задач**, соответствуют приоритетам 32…63 системы исполнения
TASKPRIO_HIGH	64…95	Высокоприоритетные задачи (например, коммуникационные задачи протоколов реального времени)
TASKPRIO_ABOVENORMAL	96…127	Задачи с приоритетом «выше среднего»
TASKPRIO_NORMAL	128…159	Задачи среднего приоритета (например, коммуникационные задачи протоколов не реального времени)
TASKPRIO_BELOWNORMAL	160…191	Задачи с приоритетом «ниже среднего»
TASKPRIO_LOW	192…223	Низкоприоритетные задачи
TASKPRIO_LOWEST TASKPRIO_IDLE TASKPRIO_MIN	224…255	Задачи, выполняемые в фоновом режиме

Приложение Б. Настройки таргет-файла для конфигурации задач

Ниже приведены настройки таргет-файла, которые определяет производитель контроллера. Эти настройки влияют на компонент **Конфигурация задач** – например, позволяют определить значения по умолчанию для его параметров.

Примеры синтаксиса для настроек приведены в OEM-документации.

Настройка	Описание
cycletimedefault	Значение по умолчанию для интервала вызова создаваемых пользователем циклических задач
cycletimemax_us	Максимально допустимое значение для интервала вызова циклических задач (в мкс)
cycletimemin_us	Минимально допустимое значение для интервала вызова циклических задач (в мкс)
defaulttaskpriority	Значение по умолчанию для приоритета создаваемых пользователем задач
externaleventcycletime	Интервал вызова задач с типом вызова Внешнее событие (по умолчанию – 0, без циклического вызова)
externalevents	Список поддержанных внешних событий
maxeventtasks	Максимально возможное число задач с типом вызова Событие
maxexternalevents	Максимально возможное число задач с типом вызова Внешнее событие
maxfreetasks	Максимально возможное число задач с типом вызова Свободное выполнение
maxintervaltasks	Максимально возможное число задач с типом вызова Циклическое
maxnumoftasks	Общее максимально возможное число задач
maxstatustasks	Максимально возможное число задач с типом вызова Статус
maxtaskpriority	Максимальное значение для приоритета задачи (то есть допустимое значение приоритета для самой низкоприоритетной задачи)
maxwatchdogsensitivity	Максимально возможная восприимчивость сторожевого таймера
mintaskpriority	Минимальное значение для приоритета задачи (то есть допустимое значение приоритета для самой высокоприоритетной задачи)
multicore-cores	Ядра CPU, к которым могут быть привязаны задачи (для CODESYS Multicore)
priorityinfo	Текст всплывающей подсказки, которая появляется в конфигурации задач при наведении курсора на значение приоритета задачи (используется для описания особенностей обработки задачи для конкретного устройства)
prohibit-duplicate-priorities	Позволяет запретить создание задач с одинаковым приоритетом
supportevent	Позволяет запретить создание задач с типом вызова Событие
supportextendedwatchdog	Позволяет запретить использование сторожевого таймера для контроля задач
supportexternal	Позволяет запретить создание задач с типом вызова Внешнее событие
supportfreewheeling	Позволяет запретить создание задач с типом вызова Свободное выполнение
supportinterval	Позволяет запретить создание задач с типом вызова Циклическое
supportmicroseconds	Определяет, в каких единицах задается интервал циклических задач – в мс или мкс
supportprofiling	Позволяет запретить мониторинг задач (соответствующая вкладка не будет отображаться в конфигурации задач)
supportstatus	Позволяет запретить создание задач с типом вызова Статус
systemtick	Дискретность системного таймера. Интервал вызова циклических задач должен быть кратен этому значению
watchdogtimemax_us	Максимальное значение для времени сторожевого таймера
default-watchdog-sensitivity	Значение по умолчанию для восприимчивости сторожевого таймера
default-watchdog-time	Значение по умолчанию для времени сторожевого таймера
watchdog-enabled	Позволяет по умолчанию активировать сторожевой таймер для создаваемых пользователем задач
defaultkindoftask	Значение по умолчанию для типа вызова задач, создаваемых пользователем

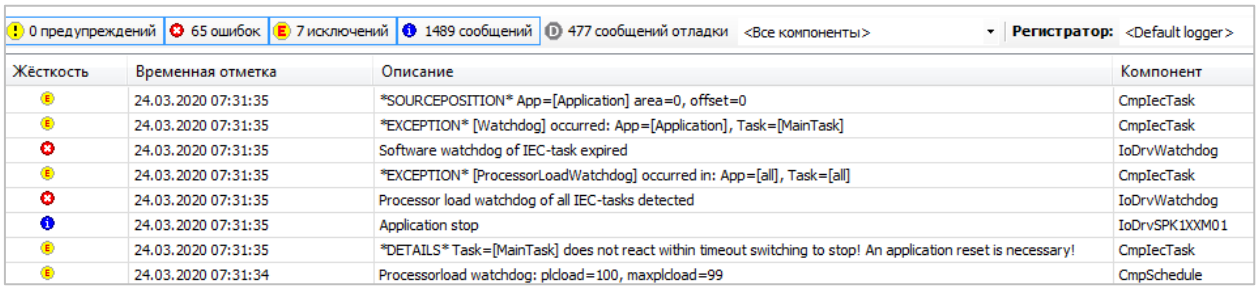
Приложение В. Настройки планировщика в конфиг-файле CODESYS

Ниже приведены возможные настройки секции [CmpShedule] конфиг-файла CODESYS (CODESYSControl.cfg), которые определяет производитель контроллера. Эти настройки влияют на работу планировщика CODESYS.

```
[CmpSchedule]
EnableLogger=1
ProcessorLoad.Maximum=80
Timeslicing.Mode=Internal
Timelicing.PlcSlicePercent=80
Timelicing.PlcSliceUs=4000
Timelicing.StartOnProcessorLoad=1
SchedulerPriority=5
SchedulerInterval=1000
DontUseMicrosecondTiming=1
```

Настройка	Описание
EnableLogger	Включить вывод сообщений компонента в журнал CODESYS
ProcessorLoad.Maximum	Максимально допустимая загрузка CPU, при превышении срабатывает сторожевой таймер (см. ниже)
Timeslicing.Mode	Режим квантования (см. в п. 2)
Timelicing.PlcSlicePercent	Количество времени, выделяемое задачам CODESYS (в % от общего)
Timelicing.PlcSliceUs	Величина одного кванта времени в микросекундах
Timelicing.StartOnProcessorLoad	Менять настройки квантования в зависимости от загрузки CPU (см. в п. 2)
SchedulerPriority	Приоритет системной задачи планировщика CODESYS
SchedulerInterval	Интервал вызова задачи планировщика CODESYS в микросекундах
DontUseMicrosecondTiming	См. примечание в п. 2
Task.Freewheeling.Cycletime	См. п. 3

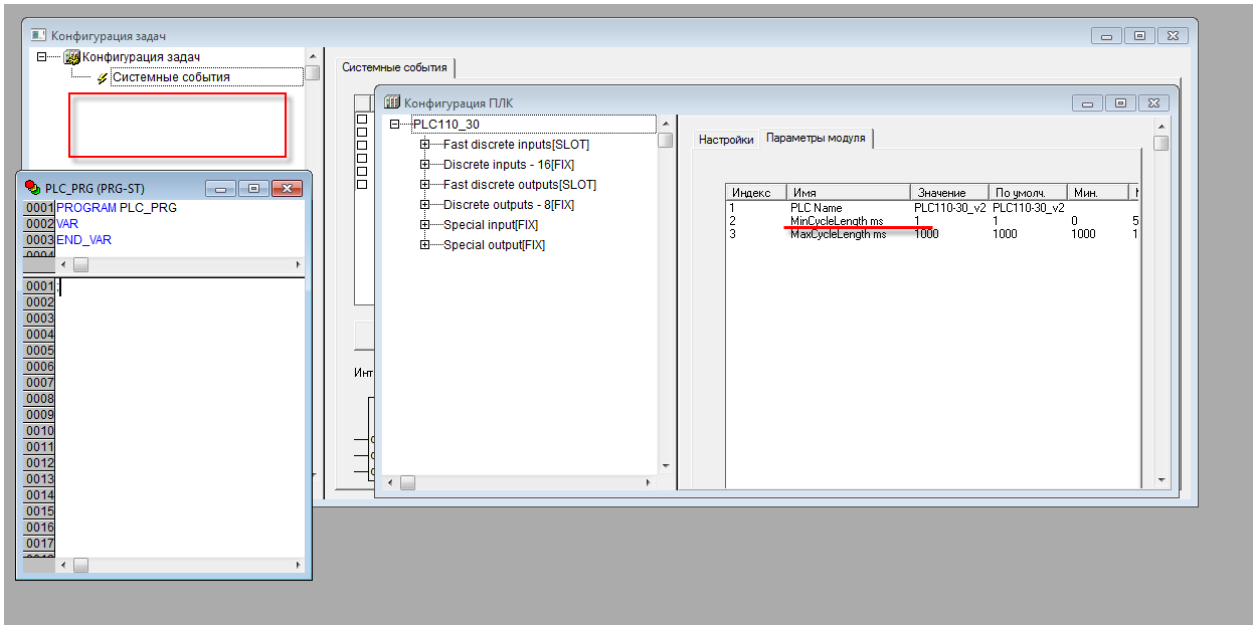
Параметр **ProcessorLoad.Maximum** влияет на работу сторожевого таймера планировщика задач. Он позволяет, например, детектировать вход приложения контроллера в бесконечный цикл. Если загрузка CPU превысит заданное значение, то будет сгенерировано исключение:



Чтобы отключить сторожевой таймер – требуется присвоить параметру значение **0**.

Приложение Г. Обработка задач в среде CoDeSys V2.3

В контроллерах ОВЕН **ПЛК1xx [M02],** программируемых в среде **CoDeSys V2.3**, реализована *кооперативная многозадачность*. По умолчанию в компоненте **Конфигурация задач** отсутствуют какие-либо задачи. В этом случае задача с названием **PLC_PRG** (она присутствует в проекте по умолчанию) циклически вызывается с интервалом, равным значению параметра **MinCycleLength**, задаваемом в **Конфигурации ПЛК** (отметим, что параметр **MaxCycleLength** не обрабатывается).



Если пользователь создает в компоненте **Конфигурация задач** свои задачи – то в случае одновременного вызова они последовательно выполняются в порядке, определяемом их приоритетами (0 – наивысший приоритет). Таким образом, каждая задача всегда выполняется от начала до конца и одна задача не может вытеснить другую.

Интервал вызова задачи не должен быть ниже, чем значение параметра **MinCycleLength**.

Дополнительная литература

1. МЭК 61131-3, МЭК 61131-8 – взгляд на задачи с позиции стандарта программирования ПЛК

2. CODESYS Control V3 Manual, RuntimeSystemOnlineHelp и другая OEM-документация – описание реализации обработки задач от разработчиков CODESYS

3. Онлайн-справка CODESYS – описание компонента **Конфигурация задач** и специфических вопросов (CODESYS Multicore, синхронизация данных между задачами)

4. Thomas Zauner. Как получить максимум от CODESYS Control Runtime System – доклад с конференции CODESYS Users Conference Russia 2015

5. Dr. Ken Ryan, Alexandria Technical College & PLCopen Board Member. Coder's Corner: The IEC 61131-3 Software Model

6. Документация Fastwel. Контроллер программируемый CPM723-01. Руководство по конфигурированию и программированию – полная, подробная и хорошо поданная информация реализации обработки задач для ПЛК Fastwel (п. 5.4.1.3, 5.4.3, 4.10.3, 4.12.2, 5.4.2 и др.)