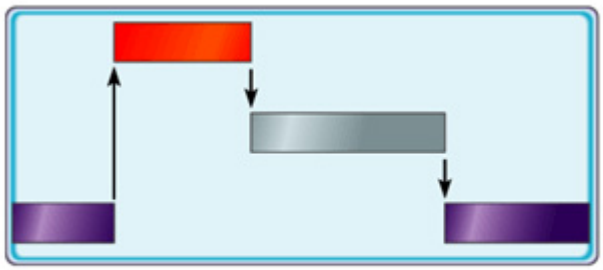
Использование задач в CODESYS V3



27.12.2023
версия 2.1

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 аппаратно-программных 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.

Документ рекомендуется читать строго последовательно.

Автор: Евгений Кислов

1. Основная информация о задачах

В стандарте МЭК 61131-3 приводится следующее определение задачи:

«*Задача – это элемент управления, который позволяет выполнять один или несколько POU на периодической или событийной основе*»

Хотя в формулировке использован термин «POU», фактически в большинстве случаев к задачам привязываются только программы. В стандарте МЭК 61131-3 упоминается возможность привязки к задачам функциональных блоков и даже функций (п. 2.7.2, пп. 7), но в стандарте МЭК 61131-8 (п. 3.12.6) эта возможность отмечена как *deprecated* (устаревшая). Далее в тексте документа задачи будут рассматриваться только как средства вызова программ (см. также описание связанных с этим нюансов CODESYS V3.5 в приложении Г).

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:

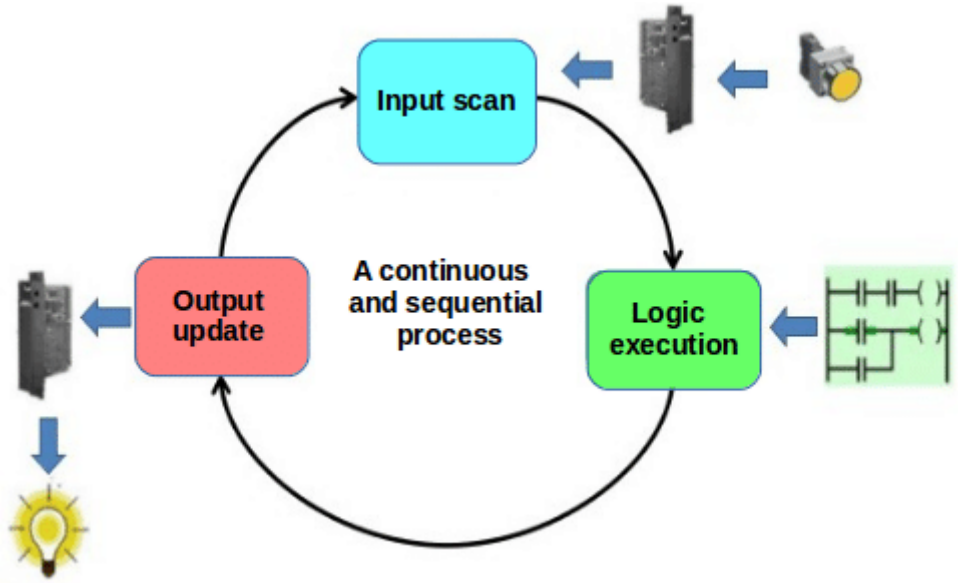


Рис. 1. Цикл работы ПЛК

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 очередной итерации этого цикла.

Подобные рисунки достаточно неплохо описывают поведение типичного однозадачного контроллера, который не имеет сетевых интерфейсов. Но большинство современных контроллеров имеют сетевые интерфейсы и позволяют создавать в рамках проекта несколько задач, что приводит нас 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. Три режима реализации управления задачами в системе исполнения CODESYS

Система исполнения (рантайм) CODESYS V3.5 предоставляет три механизма (режима) управления задачами. Разработчик устройства должен выбрать механизм, наиболее подходящий для его аппаратно-программной платформы.

2.1. Режим «однозадачность»

В режиме однозадачности (single tasking; также этот механизм часто называется *кооперативной многозадачностью*) все задачи контроллера выполняются последовательно друг за другом в бесконечном цикле. Вызов задач производится внутренним планировщиком системы исполнения 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. Серьезным недостатком данной реализации является тот факт, что коммуникационные функции также выполняются в общем цикле, поэтому проблемы с обменом (например, отсутствие ответа от опрашиваемых устройств и ожидание срабатывания таймера таймаута) могут существенно увеличить джиттер задачи (задержку вызова).

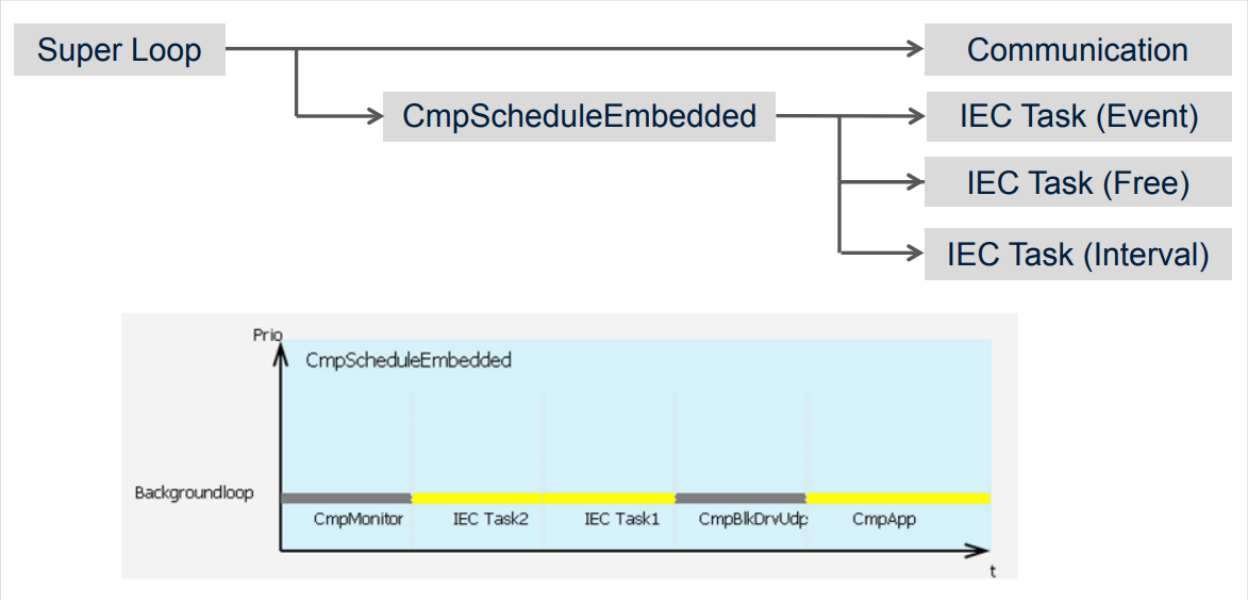


Рис. 2.2.1. Принцип обработки задач в режиме *Однозадачность*

2.2. Режим «планирование по таймеру»

В режиме 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 CODESYS в данном случае используется только для реализации программного сторожевого таймера (watchdog).

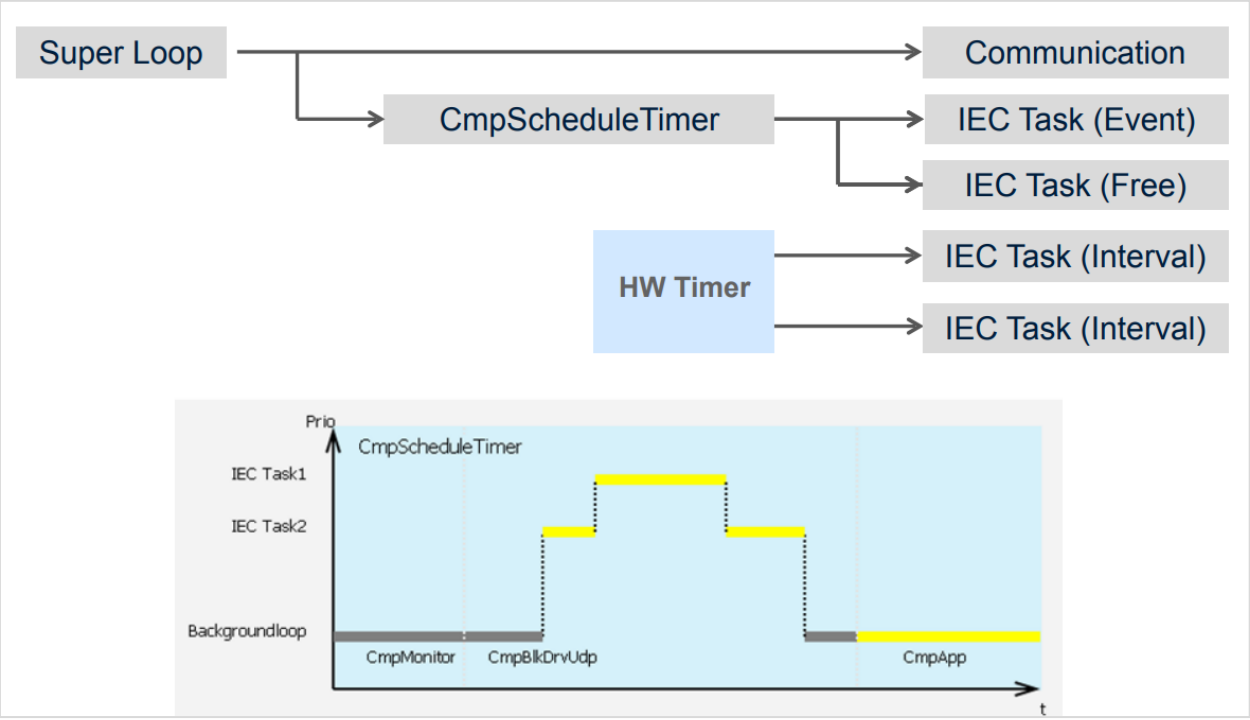


Рис. 2.2.1. Принцип обработки задач в режиме *Планирование по таймеру*

2.3. Режим «вытесняющая многозадачность». Три варианта активации задач

В режиме *вытесняющей многозадачности* (multitasking) каждая задача CODESYS соответствует задаче операционной системы.

В многозадачных ОС 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 внутренний планировщик. Этот же планировщик может выполнять функцию программного сторожевого таймера (так как не каждое устройство имеет встроенный сторожевой таймер).

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

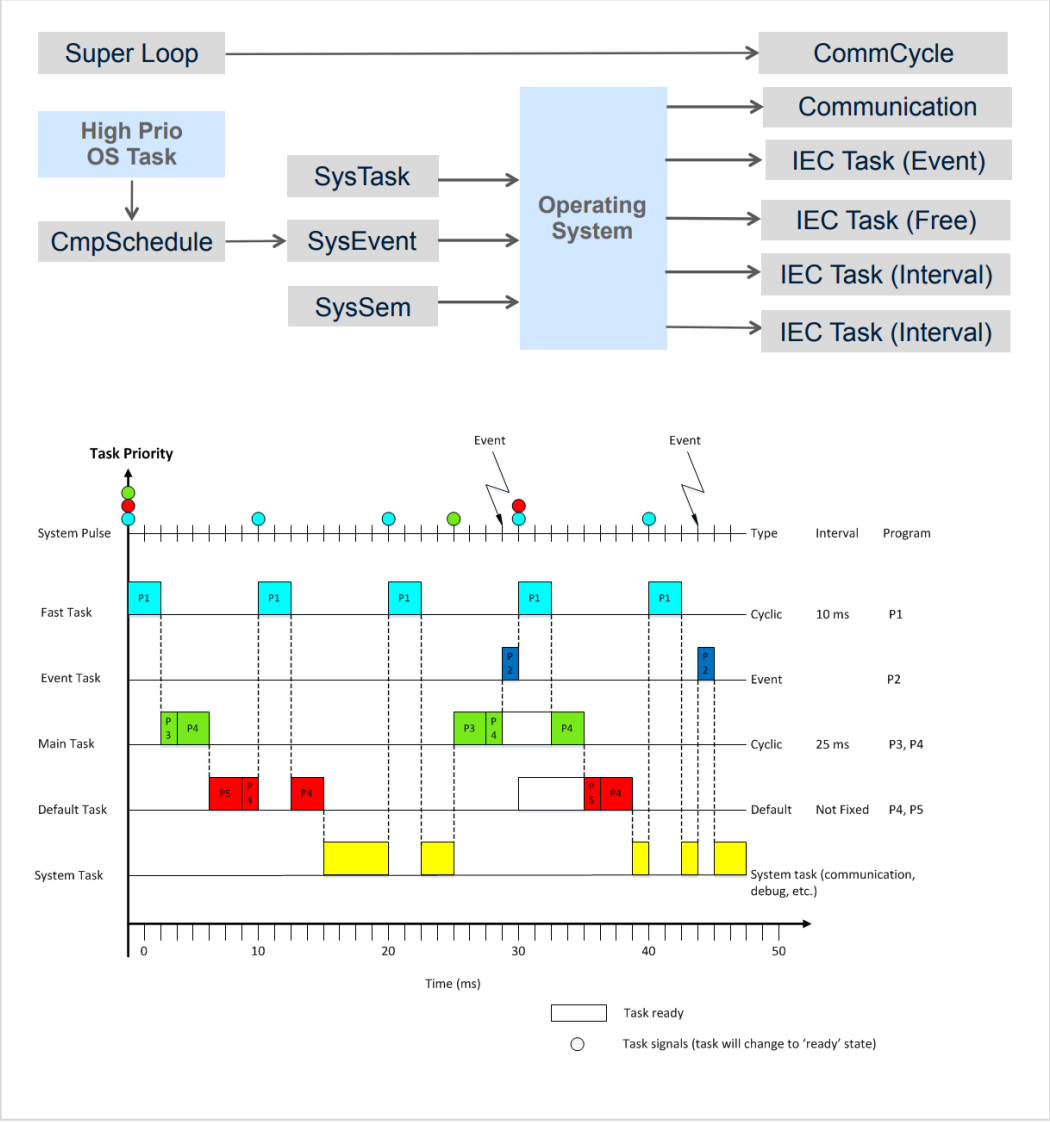


Рис. 2.3. Принцип обработки задач в режиме *Вытесняющая многозадачность* (рис. из [5])

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

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

2. Аналог варианта 1, но происходит вызов только задачи с наивысшим приоритетом. Это упрощает квантование времени (см. ниже), поскольку в каждый момент времени может исполняться только одна активная задача.

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

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 задачам CODESYS, 20% – задачам ОС).

Примечание для устройств с неточными микросекундными таймерами: планировщик задач 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 на базе ядра 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 среде разработки CODESYS выполняется в 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 циклически вызываемую задачу **MainTask**. Для корректной работы проекта CODESYS в нем должна присутствовать как минимум одна циклически вызываемая задача.

3.1. Описание компонента «Конфигурация задач»

Компонент содержит следующие вкладки:

1. Свойства

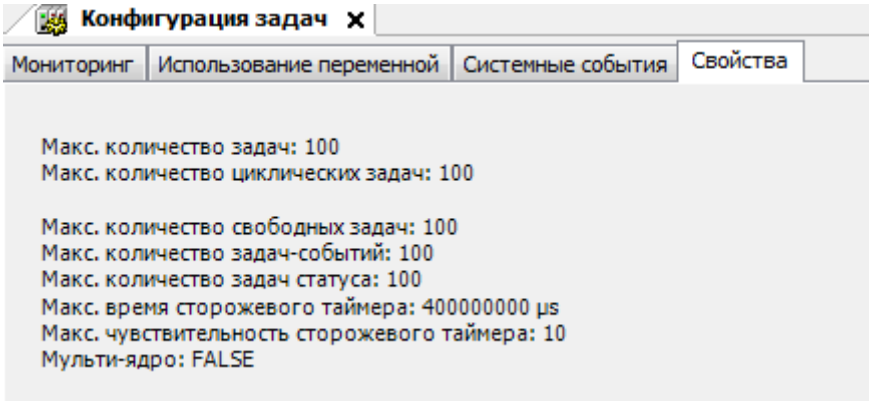


Рис. 3.1. Внешний вид вкладки **Свойства**

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

2. Мониторинг

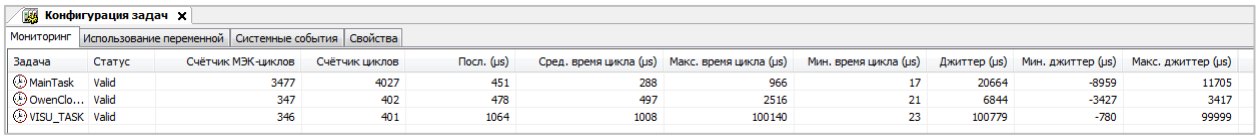


Рис. 3.2. Внешний вид вкладки **Мониторинг**

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. 3.3.

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**).

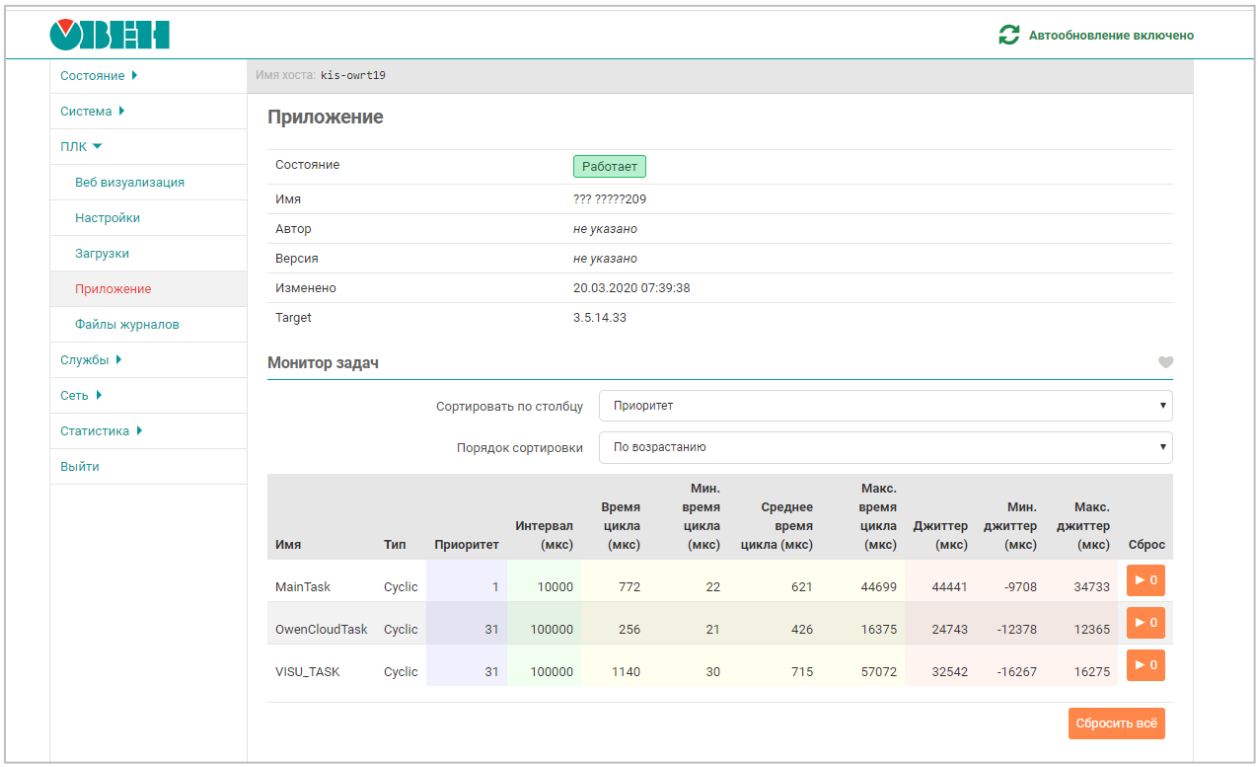


Рис. 3.3. Отображение информации мониторинга задач в web-конфигураторе для ПЛК ОВЕН

3. Использование переменной

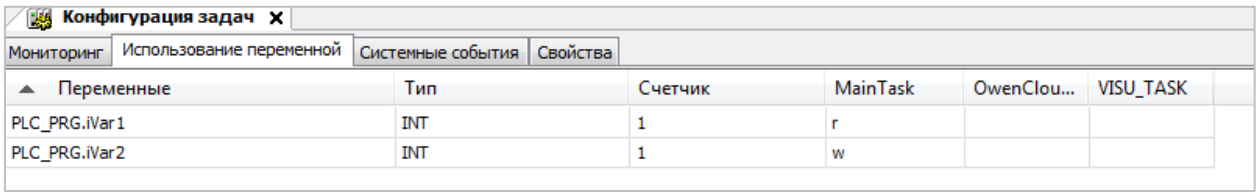


Рис. 3.4. Внешний вид вкладки **Использование переменной**

На этой вкладке выводится информация по использованию переменных в задачах проекта. Для каждой переменной отображаются задачи, в программах 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

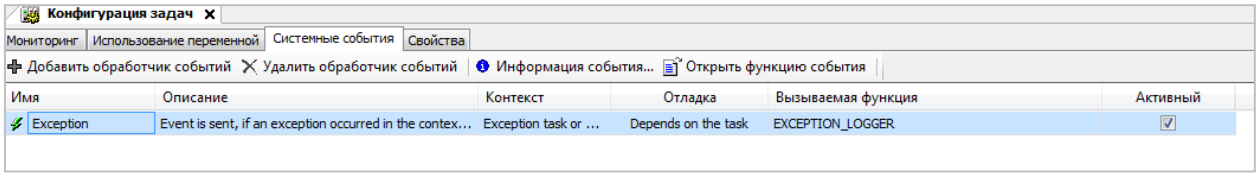


Рис. 3.5. Внешний вид вкладки **Системные события**

На этой вкладке выполняется настройка обработки системных событий. Системным событием, например, является возникновение исключения, 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.

Перехват и обработку событий в коде программы (например, это требуется в библиотеках, потому что в их состав не входит компонент **Конфигурация задач**) можно реализовать с помощью библиотеки CmpEventMgr.

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.

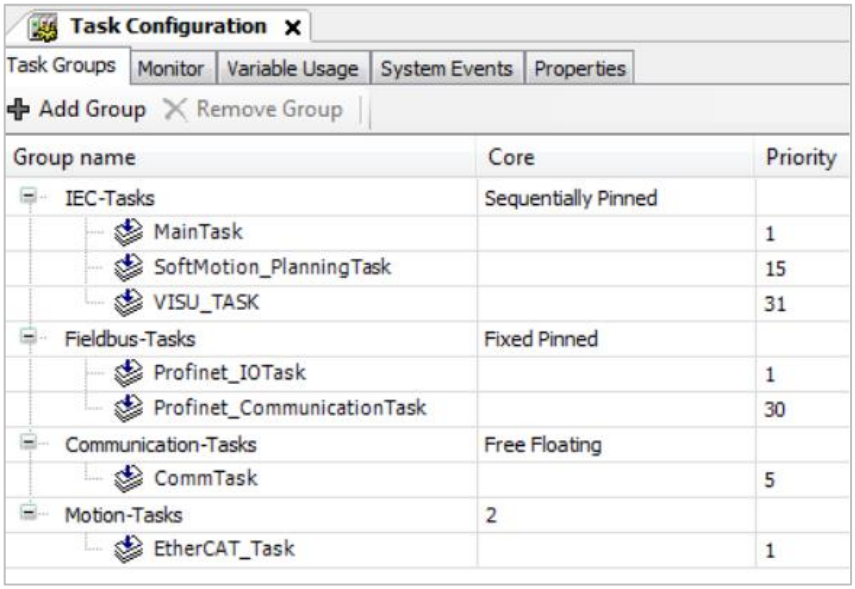


Рис. 3.6. Внешний вид вкладки **Группы задач**

6. Загрузка CPU

На этой вкладке отображаются графики загрузки ядер процессора.

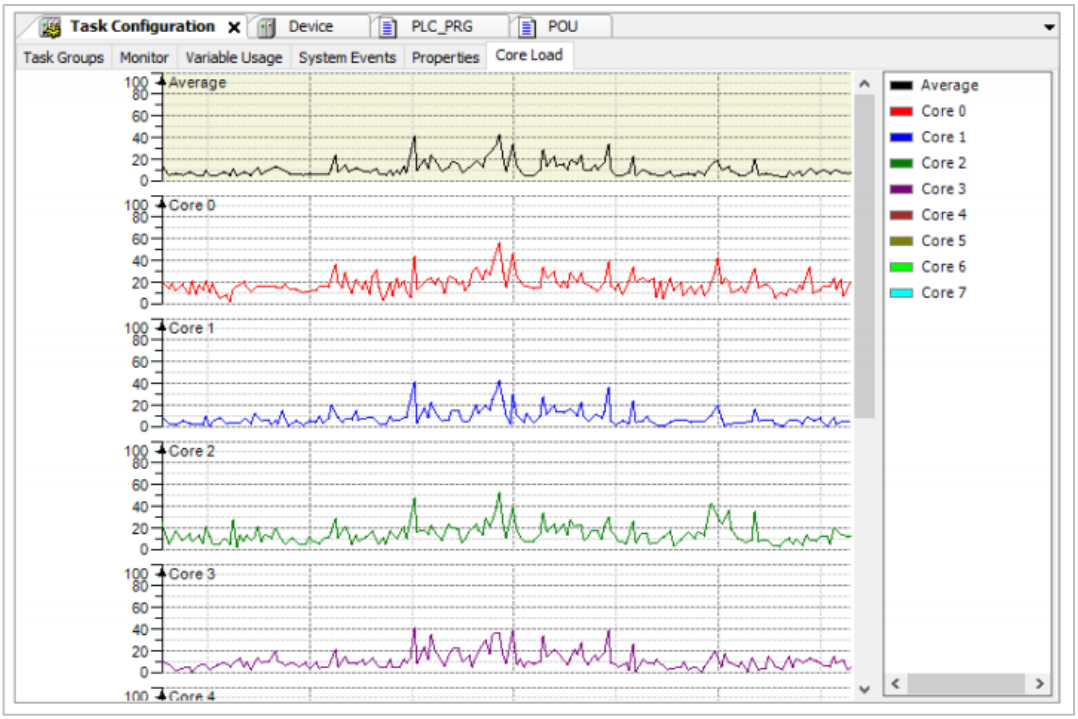


Рис. 3.7. Внешний вид вкладки **Загрузка CPU**

3.2. Описание настроек задачи

Некоторые задачи добавляются в проект автоматически при добавлении 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**.

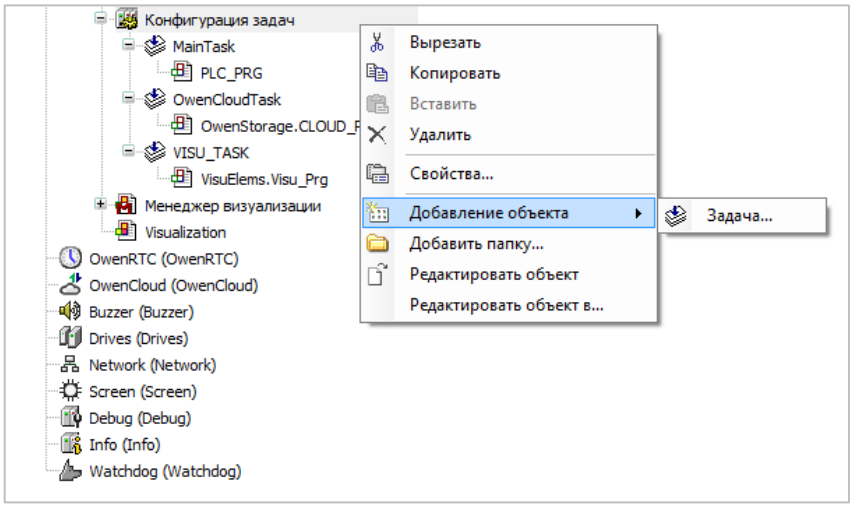


Рис. 3.8. Добавление задачи в конфигурацию задач

В верхней части окна задачи происходит работа с ее настройками. В нижней части окна происходит добавление программ[1], которые будут вызываться в данной задаче. К задаче может быть привязано неограниченное количество программ.

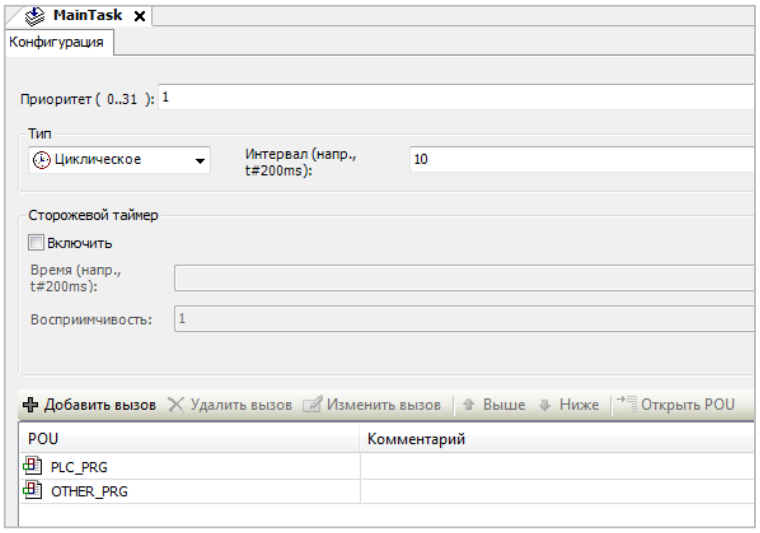


Рис. 3.9. Настройки задачи

В следующих подпунктах приведено описание настроек задачи.

[1] См. также комментарий в приложении Г.

3.2.1. Приоритет

Приоритет – это значение, которое характеризует важность данной задачи. По умолчанию оно принадлежит диапазону **0…31**, где **0** – наивысший приоритет, **31** – самый низший. Производитель контроллера может ограничить диапазон приоритетов, доступных пользователю, на уровне таргет-файла контроллера. Роль приоритета заключается в следующем: в ситуации, при которой в один и тот же момент времени выполнились условия вызова нескольких задач, будет вызвана только задача с наивысшим приоритетом.

Приоритет, который задается в настройках задачи, мы далее будет называть «МЭК-приоритетом». Кроме того, есть так называемый «рантайм-приоритет»; его диапазон – **0…255**. Этот диапазон разбит на 8 сегментов:

Табл. 3.1. Диапазон приоритетов системы исполнения

Название сегмента	**Диапазон приоритетов сегмента (BASE…END)**	**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

То есть диапазон МЭК-приоритетов задач CODESYS (**0..31**) маппируется (отображается) на диапазон рантайм-приоритетов **32…63** (REALTIME.BASE…REALTIME.END).

Границы сегментов рантайм-приоритетов в свою очередь маппируются на приоритеты операционной системы. Для ОС на базе ядра Linux с PREEMPT_RT Patch (патчем для запуска процессов ОС в реальном времени) **по умолчанию** это происходит следующим образом (дополнительная информация взята из OEM-документации CODESYS):

Табл. 3.2. Связь между различными приоритетам

Рантайм-приоритет	Приоритет потока ОС Linux / Политика планирования	МЭК-приоритет
0 (SYSTEM.BASE)	88 / SHED_FIFO	-
1…30	87…58 / SHED_FIFO	
31 (SYSTEM.END)	57 / SHED_FIFO	-
32 (REALTIME.BASE)	56 / SHED_FIFO	0
33…47	55…42 / SHED_FIFO	1…14
47 (REALTIME.BASE + 15)	41 / SHED_FIFO	15
48 (REALTIME.BASE + 16)	0 / SHED_OTHER, nice (-15)	16
49…62	0 / SHED_OTHER nice (-14…1)	17…30
63 (REALTIME.END)	0 / SHED_OTHER, nice (0)	31
>= 64 (HIGH.BASE)	0 / SHED_OTHER	-
…	0 / SHED_OTHER	-
255 (IDLE / MIN)	0 / SHED_IDLE	-

То есть диапазон МЭК-приоритетов задач проекта CODESYS делится на 2 поддиапазона:

- **0-15** – задачи реального времени с политикой планирования **SHED_FIFO**;
- **16-31** – это задачи, для которых не требуется реальное время, с политикой планирования **SHED_OTHER** и определенным nice-приоритетом.

Более подробную информацию о политиках планирования и работе планировщика Linux можно найти в сети (например, см. эту статью).

Производитель контроллеров может изменить маппирование рантайм-приоритетов на приоритеты ОС в конфиг-файле. [2]

Пример:

```
[SysTask]
; Max Linux priority(99)-OSPriority = effective Linux priority
; e.g. map IEC prio 0..15 to Linux prio 54..39 ,
OSPriority.Realtime.Base=45
OSPriority.Realtime.End=77
```

Обратите внимание, что сегменты рантайм-приоритетов не должны «перекрывать» (overlap) друг друга.

Вопрос: в чем разница между МЭК-приоритетами **0** и **15**? Особенно если предположить, что в проекте CODESYS у меня только одна задача?

Ответ: в том, как они соотносятся с другими задачами ОС. Например, в Linux обычно:

- отложенные прерывания[2] (SoftIRQ) планируются с приоритетом 50/SCHED_FIFO;
- задача CODESYS с МЭК-приоритетом 0 планируется с приоритетом 56/SCHED_FIFO;
- задача CODESYS с МЭК-приоритетом 10 планируется с приоритетом 46/SCHED_FIFO.

То есть задача с МЭК-приоритетом **0** может вытеснить обработчик программным прерываний, а задача с МЭК-приоритетом **10** – не может (и более того – может быть вытеснена этим обработчиком).

[2] По гиперссылке довольно древняя статья, со времен которой API для работы с отложенными прерываниями изменился. Актуальная информация приведена в современной литературе по архитектуре ядра Linux

3.2.2. Тип вызова

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 выше 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 и не менее 10 мс).

  2. Если в конфиг-файле CODESYS определена максимально допустимая загрузка процессора, то предоставляемое другим задачам время вычисляется по формуле:

  (100% – максимально допустимая 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, минимальное значение – 1).

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

  В этом случае свободно выполняемые задачи концептуально не будут отличаться от циклически выполняемых задач с интервалом вызова 10 мс и МЭК-приоритетом из диапазона 15…31. [7]

  Система исполнения контролирует нагрузку на процессор и гарантируется, что общая нагрузка на него от всех задач типа *свободное выполнение* не превысит 50% (каким именно способом это обеспечивается – в документации не указано).

- *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 (в контроллерах ОВЕН – не поддержан).

3.2.3. Сторожевой таймер

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 трех 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 (это называется «omitted cycle»; см. рисунок ниже);
- если циклическая задача ни разу не будет запущена на исполнение в течение времени, которое в 2 раза превышает установленный для нее интервал вызова.

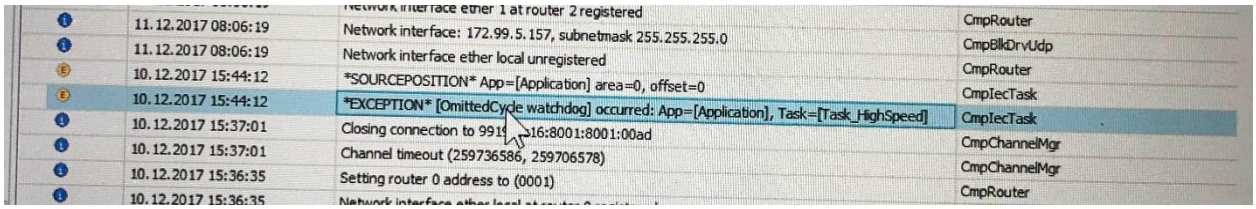


Рис. 3.10. Информация об исключении в журнале контроллера (**Device – Журнал**) при детектировании «omitted cycle»

3.2.4. Описание обработки задач для контроллеров с Linux и PREEMPT RT Patch

Планировщик задач (им может быть планировщик CODESYS или планировщик ОС, см. п. 2.3) 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 МЭК-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 еще более приоритетными задачами). 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).

Вопрос: что произойдет, если фактическое время выполнения задачи превысит интервал ее вызова? (подразумевается, что речь об единичных превышениях)

Ответ: это зависит от ее МЭК-приоритета.

- для задач реального времени (с МЭК-приоритетом 0...15): после завершения задачи она будет сразу вызвана заново. Если время выполнения задачи более чем в 2 раза превышает ее интервал вызова, то предпринимается попытка компенсировать только последний «пропущенный» цикл, а не все);
- для задач, не связанных с реальным временем (с МЭК приоритетом 16…31): не предпринимается попыток компенсировать «пропущенные» циклы. В следующий раз задача будет вызвана спустя интервал ее вызова.

С помощью настройки в конфиг-файле можно отключить для задач реального времени попытки компенсации «пропущенных циклов». [7]

```
[SysTask]
Linux.SkipLostCycles=1
```

3.3. Вкладка «Мониторинг» компонента Конфигурация задач

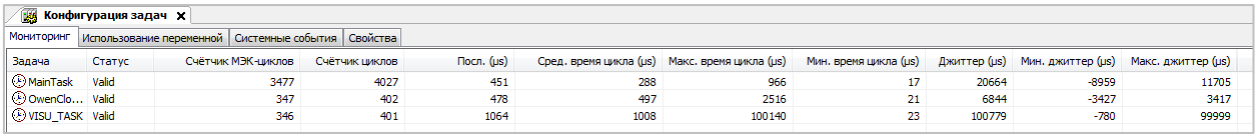


Рис. 3.11. Внешний вид вкладки **Мониторинг**

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 таблице ниже.

Табл. 3.3. 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 – например, когда контроллер находится в состоянии «СТОП»). В зависимости от конкретной реализации для конкретного устройства – значения счетчика циклов и счетчика МЭК-циклов могут как совпадать, так и отличаться.
Задан. время цикла	Интервал вызова циклической задачи, заданный в ее настройках (в **милли**секундах; но можно нажать на ячейку правой кнопкой мыши и использовать команду Показывать время цикла как мкс – в **микро**секундах, как в остальных столбцах)
Посл.	Фактические время выполнения задачи при ее последнем вызове (в микросекундах)
Сред. время цикла	Усредненное по всем произошедшим МЭК-циклам время выполнения задачи (в **микро**секундах)
Макс. время цикла	Максимальное время выполнения задачи (в **микро**секундах)
Мин. время цикла	Минимальное время выполнения задачи (в **микро**секундах)
Джиттер	Джиттер (см. ниже) задачи для ее последнего вызова (в **микро**секундах)
Мин. джиттер	Максимальное (по модулю) время опережения (см. ниже) вызова задачи (в **микро**секундах)
Макс. джиттер	Максимальное время задержки вызова задачи (в **микро**секундах)

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 моментом ее фактического вызова называется **латентностью** (latency). Разность между реальным интервалом вызова задачи и интервалом, заданным в ее настройках, называется **периодическим джиттером**

(periodic jitter). Планировщик задач может стараться компенсировать джиттер, смещая следующий вызов задачи на более ранний момент времени (т. е. вызывая ее с опережением).

Вопрос: на вкладке **Мониторинг** отображается информация о всех задачах CODESYS?

Ответ: на этой вкладке отображается информация о всех задачах приложения, добавленных в компоненте **Конфигурация задач**. Но в действительности система исполнения (процесс **codesyscontrol**) и компоненты, созданные разработчиками контроллера, создают больше потоков в рамках этого процесса. Вот пример списка потоков, созданных процессом системы исполнения CODESYS, запущенным в контроллере ОВЕН СПК1xx [M01]:

asTaskNamesList	ARRAY [0..99] OF S...	
asTaskNamesList[0]	STRING	'codesyscontrol'
asTaskNamesList[1]	STRING	'CMHooksTask'
asTaskNamesList[2]	STRING	'ubussvc_task'
asTaskNamesList[3]	STRING	'sysexec_task'
asTaskNamesList[4]	STRING	'sysexec2_task'
asTaskNamesList[5]	STRING	'buzzer_task'
asTaskNamesList[6]	STRING	'debug_task'
asTaskNamesList[7]	STRING	'drives_task'
asTaskNamesList[8]	STRING	'modem_task'
asTaskNamesList[9]	STRING	'modem2_task'
asTaskNamesList[10]	STRING	'owencloud_task'
asTaskNamesList[11]	STRING	'owenrtc_task'
asTaskNamesList[12]	STRING	'network_task'
asTaskNamesList[13]	STRING	'spk1xxm01_task'
asTaskNamesList[14]	STRING	'screen_task'
asTaskNamesList[15]	STRING	'watchdog_task'
asTaskNamesList[16]	STRING	'CMHooksTask'
asTaskNamesList[17]	STRING	'SysWindowSingle'
asTaskNamesList[18]	STRING	'CAAEventTask'
asTaskNamesList[19]	STRING	'SchedProcessorL'
asTaskNamesList[20]	STRING	'SchedException'
asTaskNamesList[21]	STRING	'Schedule'
asTaskNamesList[22]	STRING	'TaskGapTask'
asTaskNamesList[23]	STRING	'OPCUAServerWork'
asTaskNamesList[24]	STRING	'OPCUAServerWork'
asTaskNamesList[25]	STRING	'OPCUAServerSche'
asTaskNamesList[26]	STRING	'WebServerCloseC'
asTaskNamesList[27]	STRING	'BlkDrvTcp'
asTaskNamesList[28]	STRING	'BlkDrvUdp'
asTaskNamesList[29]	STRING	'OPCUAServer'
asTaskNamesList[30]	STRING	'CMCommCycleTask'
asTaskNamesList[31]	STRING	'Async_AppServic'
asTaskNamesList[32]	STRING	'AsyncTask128'
asTaskNamesList[33]	STRING	'IoMgrDiagTask'
asTaskNamesList[34]	STRING	'MainTask'
asTaskNamesList[35]	STRING	'OwenCloudTask'
asTaskNamesList[36]	STRING	'VISU_TASK'

Рис. 3.12. Пример списка потоков процесса системы исполнения CODESYS для контроллеров ОВЕН

Из них только три (**MainTask**, **OwenCloudTask**, **VISU_TASK**) добавлены в проекте в компоненте **Конфигурация задач**, а остальные создаются «под капотом» – самой системой исполнения (например, **CMHooksTask**, **SysWindowSingleTask** и т. д.) или компонентами, разработанными ОВЕН (**ubussvc_task**, **sysexec_task** и т. д.).

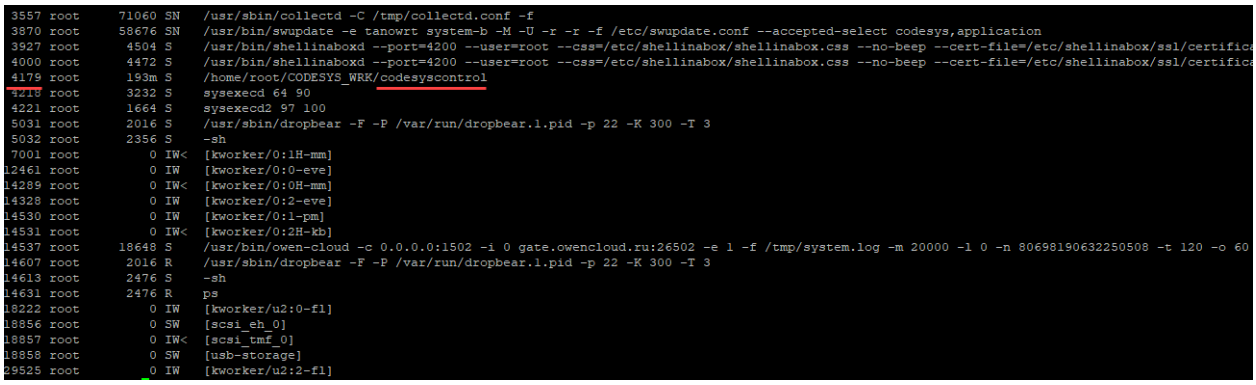
Вопрос: как получить информацию о потоках процесса системы исполнения на моем конкретном контроллере?

Ответ:

1. Подключитесь к контроллеру по SSH (если он это поддерживает). Для контроллеров ОВЕН используется логин **root** и пароль по умолчанию **owen**. Если у вас другой контроллер – запросите информацию о логине и пароле для **SSH** у его производителя.

2. Введите команду **ps** для отображения списка процессов (в ряде случае вы не увидите в выводе нужного процесса – тогда используйте команду **ps aux**, **top** или другую из известных вам).

Найдите в выводе процесс **codesyscontrol**. Запишите его идентификатор (**PID**) – на скриншоте ниже он равен **4179**:



3. Введите команду **ls /proc/<PID>/task**, где вместо <PID> подставьте полученное в пп. 2 значение. Вы получите список дескрипторов потоков процесса (**THREAD_IDs**):

4. Используйте команду **cat /proc/<PID>/task/<THREAD_ID>/stat**, чтобы получить информацию о конкретном потоке. Расшифровка полей приведена здесь. Отображаемое имя потока ограничивается его первыми 15 символами.

4. Задачи и компоненты дерева проекта

Пользователь CODESYS может добавлять в дерево проекта (**ПКМ** на узел **Device** – **Добавить устройство**) различные компоненты – например, компоненты протоколов обмена, архиватор и т. д. Добавление некоторых компонентов (например, компонентов протоколов EtherCAT и Profinet) приводит к автоматическому созданию соответствующих им задач.

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

На самом деле, словосочетание «вызов компонента» не совсем корректно. Правильнее говорить о «вызове методов экземпляра функционального блока компонента». Этот экземпляр неявно создается при добавлении компонента в дерево проекта; его имя совпадает с именем объекта в дереве. Каждый компонент реализует интерфейс ICmpIoDrv из библиотеки **IoDriver Interfaces**. Именно в нем описываются упомянутые выше методы. Нас интересуют три из них:

- **IoDrvReadInputs** – в этом методе выполняется передача значений из функционального блока компонента в его входные каналы вкладки **Соотнесение входов-выходов**;
- **IoDrvWriteOutputs** – в этом методе выполняется передача значений из выходных каналов вкладки **Соотнесение входов-выходов** компонента в переменные его функционального блока;
- **IoDrvStartBusCycle** – в этом методе выполняется бизнес-логика компонента.

Давайте разберемся, в контексте каких задач вызываются эти методы.

Метод **IoDrvStartBusCycle** вызывается в **задаче цикла шины** компонента – мы поговорим о том, где она выбирается, на следующей странице. По умолчанию этот метод вызывается после вызова метода **IoDrvWriteOutputs** компонента, но у разработчика компонента существует возможность перенести его вызов перед вызовом метода **IoDrvReadInputs**.

Задача, в которой вызываются методы **IoDrvReadInputs/IoDrvWriteOutputs**, зависит от значения параметра **Всегда обновлять переменные**, устанавливаемого в конфигураторе компонента на вкладке **Соотнесение входов-выходов**.

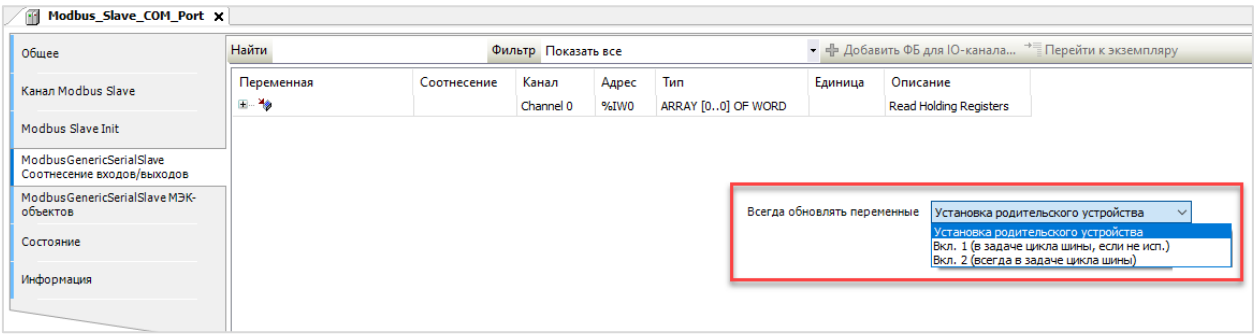


Рис. 4.1. Установка значения параметра **Всегда обновлять переменные**

- **Установка родительского устройства** – в качестве задачи вызова методов используется задача цикла шины родительского компонента. Если в нем тоже используется родительская установка – то используется задача цикла шины его родительского компонента и т. д. вплоть до узла **Device**.

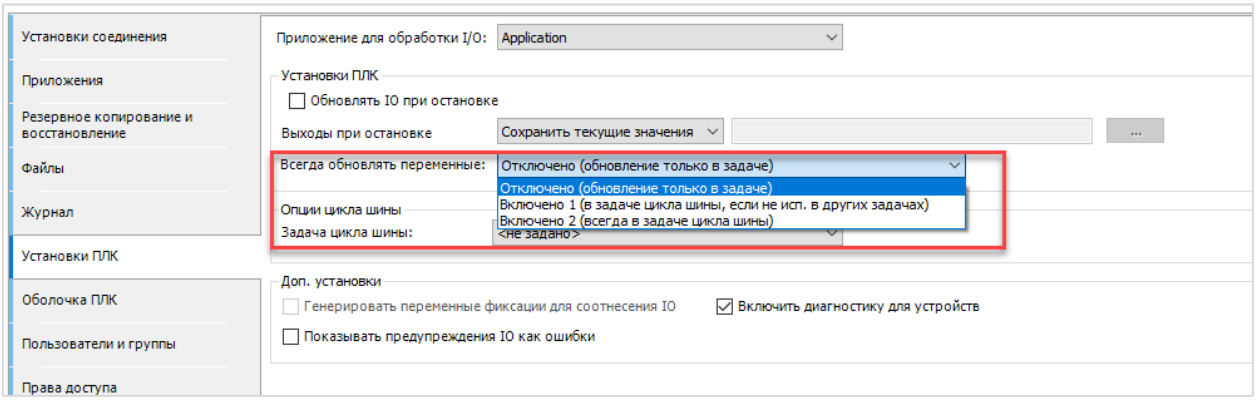


Рис. 4.2. Выбор задачи методов **IoDrvReadInputs/IoDrvWriteOutputs**

Если параметр будет установлен в значение **Отключено (обновление только в задаче)**, то вызов метода **IoDrvReadInputs** будет выполняться в контексте **всех задач**, переменные которых привязаны ко входам компонента и используются в коде программ этих задач, а вызов метода **IoDrWriteOutputs** – в контексте **всех задач**, переменные которых привязаны к его выходам и используются в коде программ этих задач.

- **Включено 1 (в задаче цикла шины, если не исп. в других задачах)** – если переменные, привязанные к входам/выходам компонента, не используются в контексте задач, к которым привязаны программы, в которых объявлены эти переменные – то вызов методов **IoDrvReadInputs/IoDrvWriteOutputs** происходит в задаче цикла шины компоненты. Если переменные используются – то вызов методов происходит в контексте всех этих задач (как для предыдущего варианта).
- **Включено 2 (всегда в задаче цикла шины)** – вызов **IoDrvReadInputs/IoDrvWriteOutputs** всегда выполняется в задаче цикла шины компонента.

Выше уже множество раз была упомянута «**задача цикла шины**», в которой выполняется бизнес-логика компонента. Пора бы уже показать, где именно она выбирается.

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

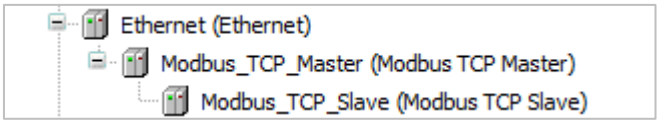


Рис. 4.3. Настройка контроллера в режиме **Modbus TCP Master**

В компоненте **Modbus TCP Master** на вкладке **Modbus TCP Master Соотнесение входов/выходов** можно выбрать конкретную задачу, в цикле которой будет вызываться методы экземпляра функционального блока **Modbus_TCP_Master**.

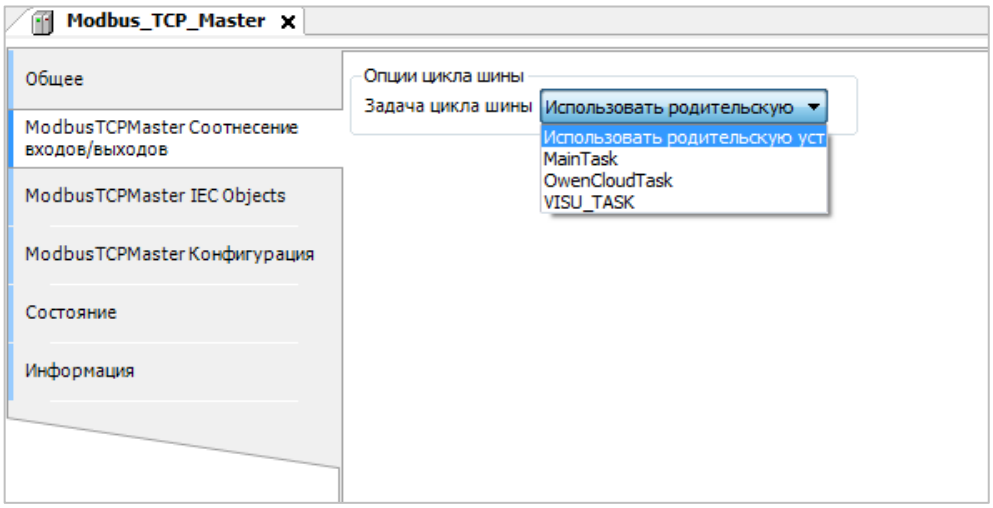


Рис. 4.4. Выбор задачи цикла шины для компонента **Modbus TCP Master**

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 **Etherne**t (см. рис. 4.3).

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

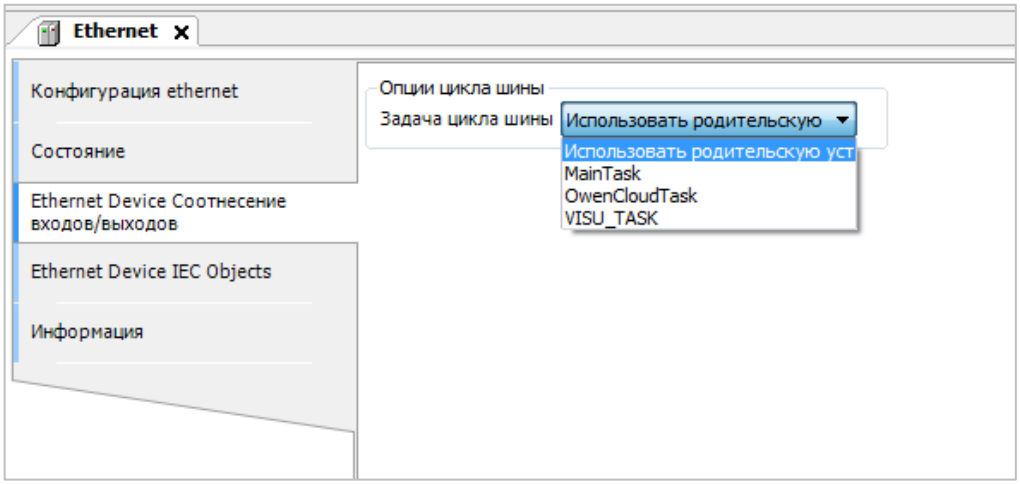


Рис. 4.5. Выбор задачи цикла шины для компонента **Ethernet**

Как мы видим, в этом компоненте тоже по умолчанию 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:

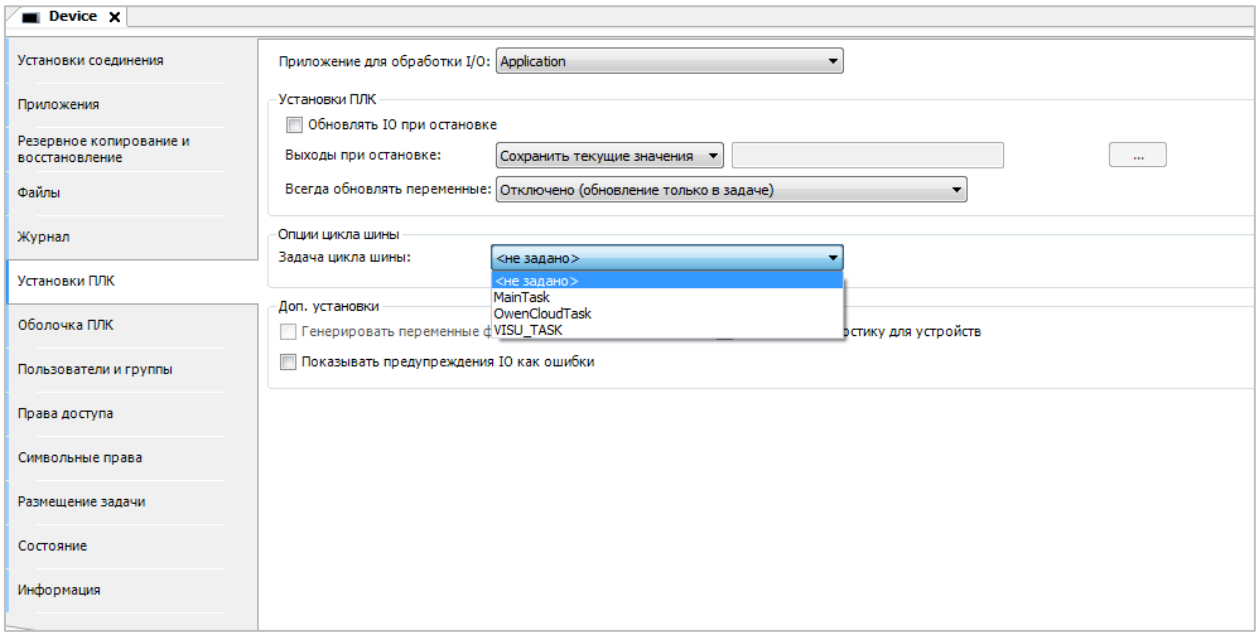


Рис. 4.6. Выбор задачи цикла шины в узле **Device**

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

Разработчик компонента может повлиять на это поведение:

- с помощью специальной настройки он может сделать так, чтобы по умолчанию в качестве задачи цикла шины компонента использовалась задача проекта с **наибольшим** интервалом вызова;
- с помощью специальной настройки он может сделать так, чтобы задачей цикла шины компонента являлась конкретная задача проекта – при этом можно как разрешить пользователю ее изменить, так и запретить (во втором случае на рис. 4.4 не было бы настройки **Опции цикла шины**);
- с помощью специальной настройки он может сделать так, чтобы при добавлении компонента в проект автоматически создавалась бы новая задача – обычно именно она и будет являться задачей цикла шины компонента (см., например, компоненты **EtherCAT Master**, **OwenArchiver** и др.).

Необходимо отметить, что бизнес-логика компонента не обязана выполняться именно в методе **IoDrvStartBusCycle**. Например, для каждого компонента таргет-файла контроллеров ОВЕН (**OwenRTC**, **Buzzer** и т. д.), которые реализованы на языке ANSI C, создается фоновый поток, в котором выполняется их бизнес-логика (их видно на рис. 3.12 – **owenrtc_task**, **buzzer_task** и т. д), а в задаче цикла шины (для этих компонентов задача цикла шина установлена по умолчанию – как задача проекта с наименьшим интервалом вызова) выполняются лишь вызовы методов **IoDrvReadInputs** и **IoDrvWriteOutputs** для синхронизации значений каналов компонента и переменных экземпляра его функционального блока.

Вернемся к нашему примеру с **Modbus TCP Master**.

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 обмен будет работать корректно.

С другой стороны, если пользователь, например, увеличит время задачи **MainTask** до 500 мс (и при этом в проекте не будет задач с меньшим временем цикла) – то, вероятнее всего, обмен будет работать «медленно» и «некорректно» – особенно если в компоненте **Modbus TCP Slave**[3] в настройках Modbus-запросов установлен интервал их отправки по умолчанию (100 мс). То есть, с одной стороны, пользователь хочет, чтобы запросы отправлялись «часто», но сам код логики их отправки, получения и разбора ответа и синхронизации значений регистров с каналами вкладки **Соотнесение входов-выходов** будет выполняться «редко» – естественно, ничего хорошего из этого не выйдет.

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

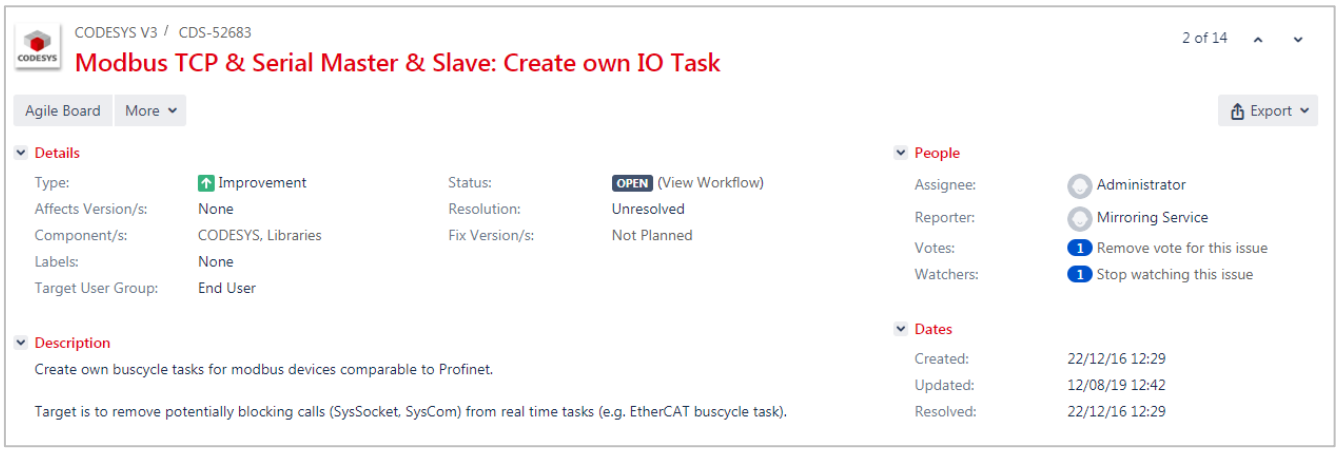


Рис. 4.7. Пожелание по автоматическому созданию задачи для компонентов Modbus в баг-трекере CODESYS

[3] Раньше в тексте этот компонент не упоминался, потому что для него нельзя выбрать задачу цикла шины – он «наследует» ее от компонента **Modbus TCP Master**.

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**.

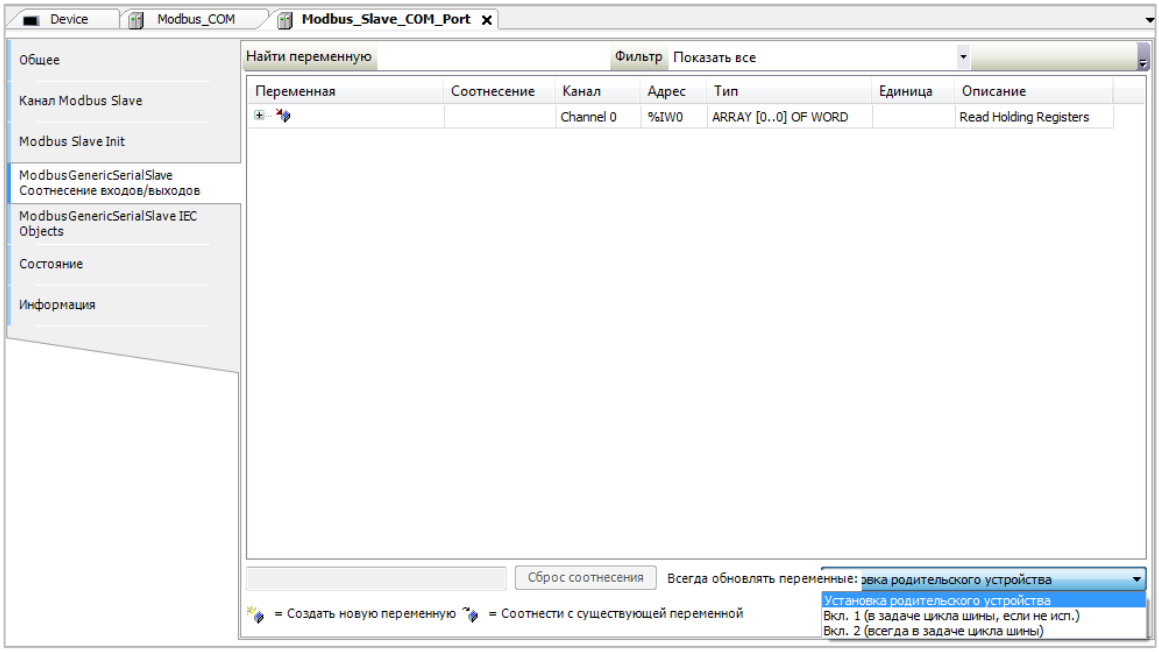


Рис. 4.8. Настройка **Всегда обновлять переменные**

Этот параметр определяет, когда происходит обновление данных в каналах компонента (т. е. синхронизация значений каналов с переменным экземпляра функционального блока компонента), к которым привязываются переменные проекта. Его возможные значения:

- *Установка родительского устройства* – для каналов с привязанными переменными обновление происходит в задаче цикла шины 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 компонента. 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 их обработки (или к каналам компонентов вообще еще ничего не привязано), у пользователей часто возникает проблема – несмотря на отсутствие каких-то ошибок обмена каналы имеют значение «0» и выделяются серым цветом[4] (даже если фактически в них должны быть какие-то считанные значения).

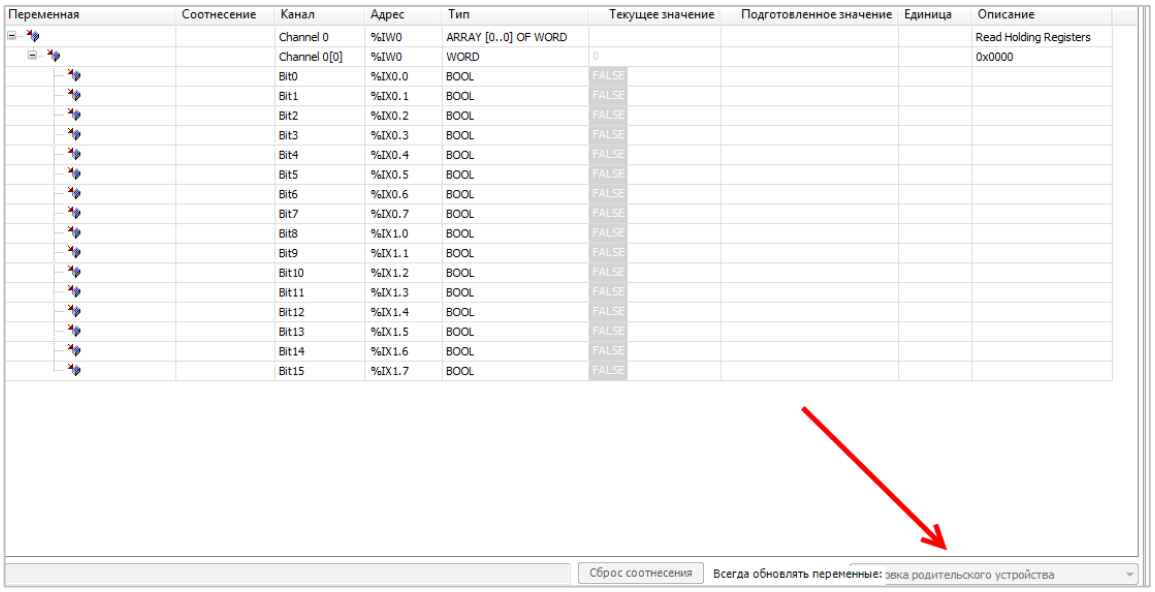


Рис. 4.9. Внешний вид необновляемых каналов

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

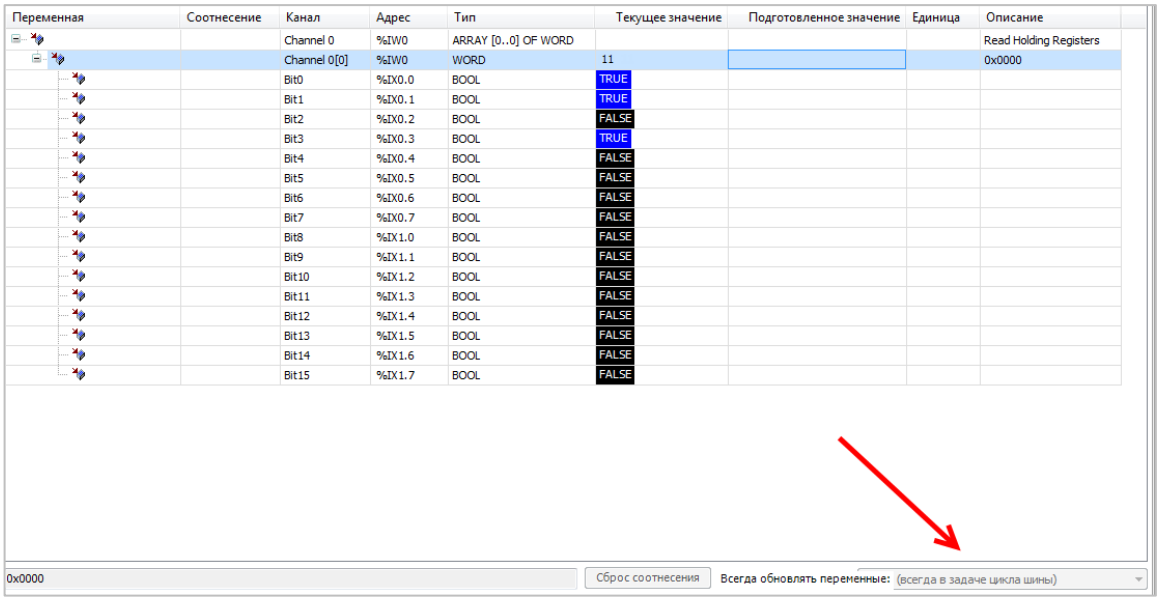


Рис. 4.10. Внешний вид обновляемых каналов

[4] Начиная с версии CODESYS V3.5 SP17 индикацию таких «неопрашиваемых каналов» сделали более явной.

Напоследок стоит привести несколько рисунков из документации CODESYS, поясняющих принцип работы коммуникационных компонентов (где Bus cycle task – это задача цикла шины).

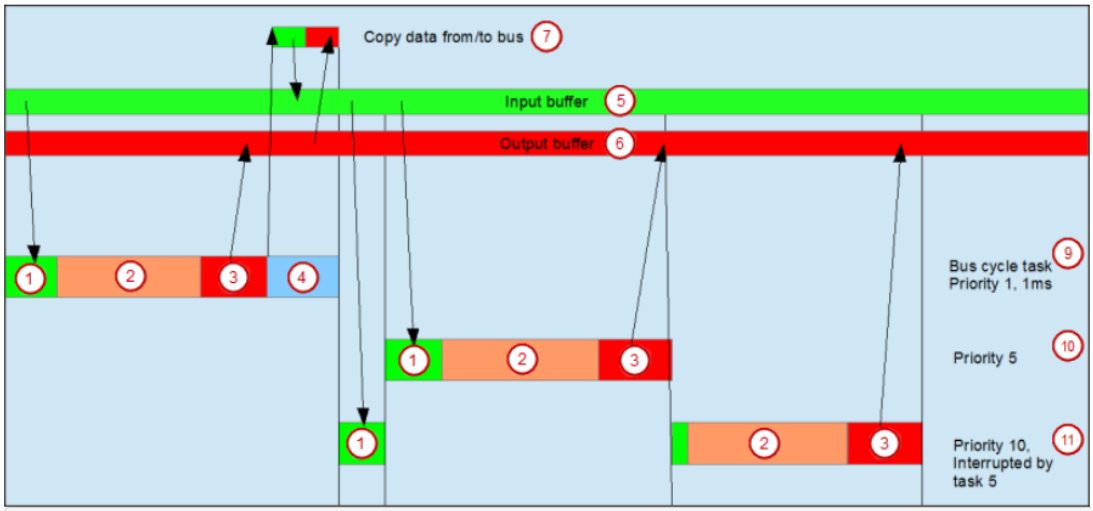


Рис. 4.11. Принцип работы коммуникационных компонентов (1)

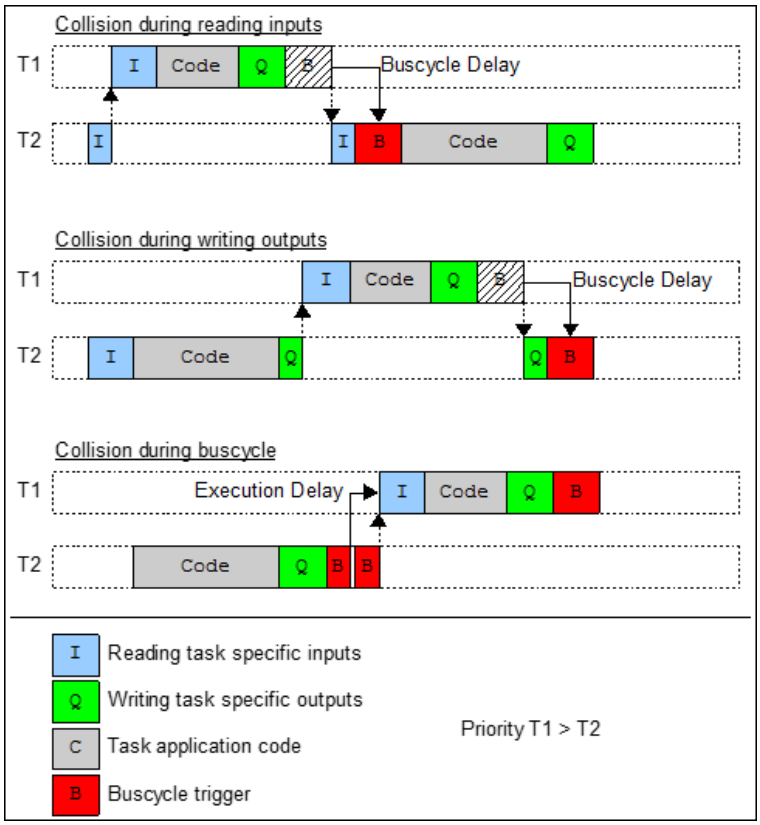


Рис. 4.12. Принцип работы коммуникационных компонентов (2)

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. рис. 3.6).

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

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 более 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). Надо отметить, что он поддерживается не всеми контроллерами. Также информация по синхронизации данных между задачами приведена в этом разделе онлайн-справки. Еще один вариант использование семафоров (см. библиотеку SysSem).

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.

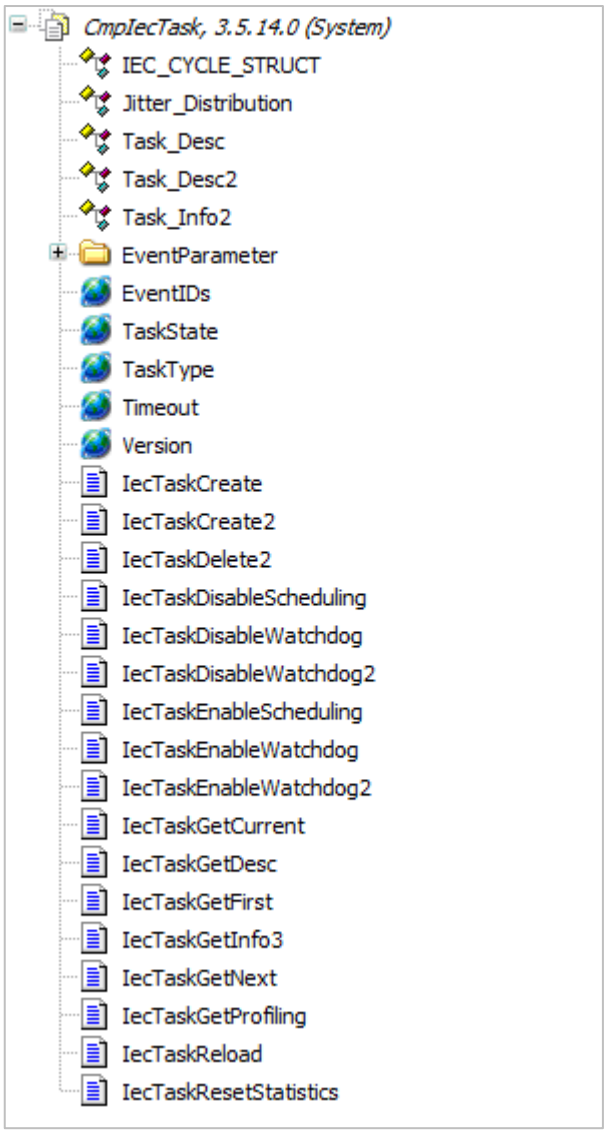


Рис. 7.1. Состав библиотеки **CmpIecTask**

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 вызова **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 задачу цикла шины (за исключением случая, когда вы четко понимаете, зачем вы это делаете и как именно это отразится на вашем проекте).
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 A. Настройки таргет-файла для конфигурации задач

Ниже приведены настройки таргет-файла, которые определяет производитель контроллера. Эти настройки влияют на компонент **Конфигурация задач** – например, они 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 выполнение**
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 выполнение**
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 сторожевого таймер для создаваемых пользователем задач
defaultkindoftask	Значение по умолчанию для типа вызова задач, создаваемых пользователем

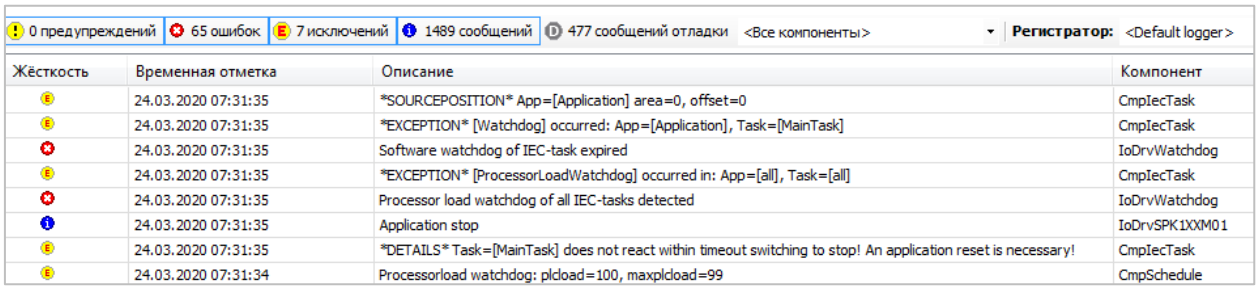
Приложение Б. Настройки планировщика в конфиг-файле CODESYS

Ниже приведены некоторые возможные настройки секции **[CmpShedule]** конфиг-файла системы исполнения CODESYS (**CODESYSControl.cfg**), которые определяет производитель контроллера. Эти настройки влияют на работу планировщика задач 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. 2.3)
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. 2.3)
SchedulerPriority	Приоритет системной задачи планировщика CODESYS
SchedulerInterval	Интервал вызова задачи планировщика CODESYS в микросекундах
DontUseMicrosecondTiming	См. примечание в п. 2.3
Task.Freewheeling.Cycletime	См. п. 3.2.2

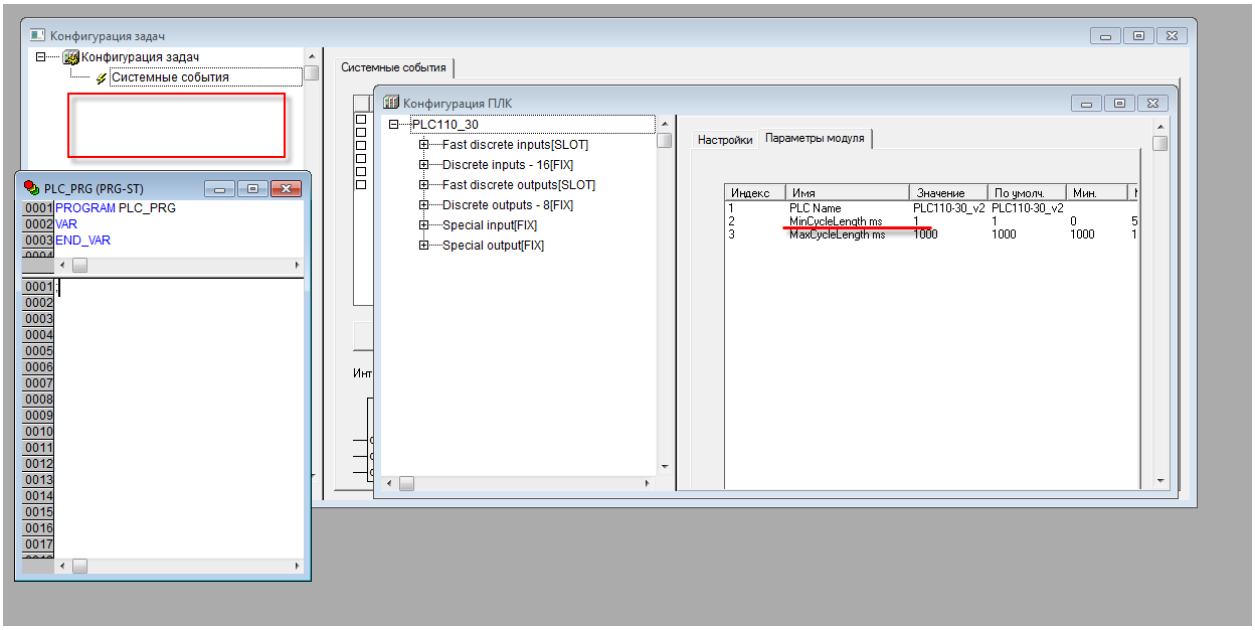
Параметр **ProcessorLoad.Maximum** влияет на работу сторожевого таймера планировщика задач. Он позволяет, например, детектировать вход приложения контроллера в бесконечный цикл. Если загрузка CPU превысит заданное значение, то будет сгенерировано исключение:



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

Приложение В. 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 и был запланирован «на будущее»).



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 и до конца», и одна задача не может вытеснить другую.

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

Приложение Г. Какие типы POU можно привязать к задаче в CODESYS V3.5?

В рамках документа мы рассматривали задачи как средство вызова программ. Действительно, если в настройках задачи нажать кнопку **Добавить вызов** – то в появившемся списке вы увидите только программы проекта.

Но, строго говоря, это лишь ограничение графического интерфейса среды разработки. Вы можете привязать к задаче несколько программ, а далее нажать на любую из них правой кнопкой мыши, выбрать пункт **Свойства** и изменить имя привязанного к задаче POU – в том числе, на имя:

- функции;
- экземпляра функционального блока;
- метода любого типа POU;
- действия любого типа POU.

В последних трех случаях в имени должны быть указаны все необходимые пространства имен (например, имя программы или списка глобальных переменных, в котором объявлен экземпляр функционального блока).

Эта возможность используется некоторыми компонентами самого CODESYS (например, **TrendRecordingManager** автоматически создает задачу, к которой привязан вызов метода **CyclicCall** экземпляра функционального блока **g_TrendRecordingManager**) – но не подразумевается, что ее должны эксплуатировать конечные пользователи. Привязка к задачам любых типов POU, за исключением программ, затрудняет понимание потока управления проекта и его отладку.

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.)

7. Codesys Taskkonfiguration – FAQ