И.В. ПЕТРОВ (Компания “Пролог”), В. ДОЛЛ (Компания “3S-Smart Software Solutions GmbH”)

Отладка прикладных ПЛК программ в CoDeSys (часть 10)

Необходимость в прикладных программах ПЛК выполнять множество различных операций одновременно приводит к необходимости распараллеливания вычислений. В самых простых случаях мы организуем параллельные операции самостоятельно в одной программе, например, в языке SFC с помощью параллельных ветвей. Для более сложных и объемных вычислений оказывается удобнее создать многозадачный проект.

Под многозадачностью понимается попеременное выполнение нескольких задач на одном процессоре. Распределением процессорного времени между задачами занимается система исполнения. В результате мы избавляемся от написания вспомогательного кода для организации параллельных операций в прикладной программе. Она получается компактнее, понятнее и быстрее. Самое важное, что мы достигаем необходимую реактивность системы, то есть требуемое для каждой задач время реакции на изменение входов контроллера.

Естественно, реально достижимое время реакции ограничено быстродействием контроллера. При создании многозадачного проекта совсем несложно сделать так, что задачи начнут мешать друг другу вплоть до полного краха системы и абсолютно непонятных эффектов при отладке. Все начинающие пользователи так и делают. Поэтому если вы можете обойтись без использования задач в своем проекте, то не используйте их.

Создание надежно работающих многозадачных приложений требует четкого понимания того, как устроен механизм выполнения задач. Это особенно важно при отладке. В этой статье мы постараемся рассмотреть устройство и тонкости реализации многозадачности в CoDeSys, так чтобы это было понятно пользователям, не имеющим достаточного опыта программирования.

С июня этого года в CoDeSys появилась поддержка интерфейса среды программирования на русском языке. Поэтому в дальнейшем в иллюстрациях будет использоваться русская версия.

Задачи и контроль времени выполнения

Слово '**задача**' в CoDeSys служит исключительно для организации планирования вычислительного процесса. На практике мы имеем дело с задачами только в 'Конфигураторе задач'. Здесь мы создаем задачи, назначаем им имена и настраиваем их параметры. Для выполнения полезной работы с каждой задачей связывается как минимум одна программа (POU типа PROGRAM). Когда мы создаем новый проект в CoDeSys, то в нем неявно организуется одна задача, содержащая вызов одной программы PLC_PRG.

Данная задача всегда выполняется циклически. В зависимости от типа ПЛК могут быть два способа ее вызова. Обычно она запускается так часто как это только можно. Как только программа отрабатывает один цикл вызова, система исполнения выполняет обслуживание входов/выходов и снова запускает прикладную задачу. Очевидно, период вызова такой задачи не стабилен и может плавать в зависимости от того, по какой ветви работает программа в данный момент. Обслуживание системой исполнения аппаратных устройств, коммуникационных каналов, работа отладчика также могут влиять на периодичность такой задачи. В окне 'Конфигурация ПЛК' задача такого типа называется '**свободная**' (freewheeling). Второй способ состоит в поддержке стабильного интервала выполнения единственной задачи. Даже если программа пуста, то она не будет вызываться чаще, чем задано. В окне 'Конфигурация ПЛК' такая задача называется '**циклическая**'.

Например, в ПЛК-150 компании Овен по умолчанию единственная задача вызывается циклически с интервалом 1 мс. Время рабочего цикла можно изменять от 0 до 50 мс (параметр MinCycleLength в конфигурации ПЛК). При задании нулевого времени контроль отключается и задача превращается в 'свободную'. Такой встроенный механизм настройки ПЛК достаточно удобен для начинающих пользователей.

Для любых типов ПЛК мы можем открыть в CoDeSys диалоговое окно 'Конфигурация ПЛК', создать новую задачу и настроить ее параметры так, как нам нужно. Обратите внимание на то, что если мы беремся за самостоятельное определение задач в CoDeSys и создаем в конфигураторе хотя бы одну задачу, то действовавший по умолчанию вызов PLC_PRG отключается. Нам нужно будет вручную добавить вызов этой программы в одну из задач, даже если она всего одна. *См. рис. 1.*

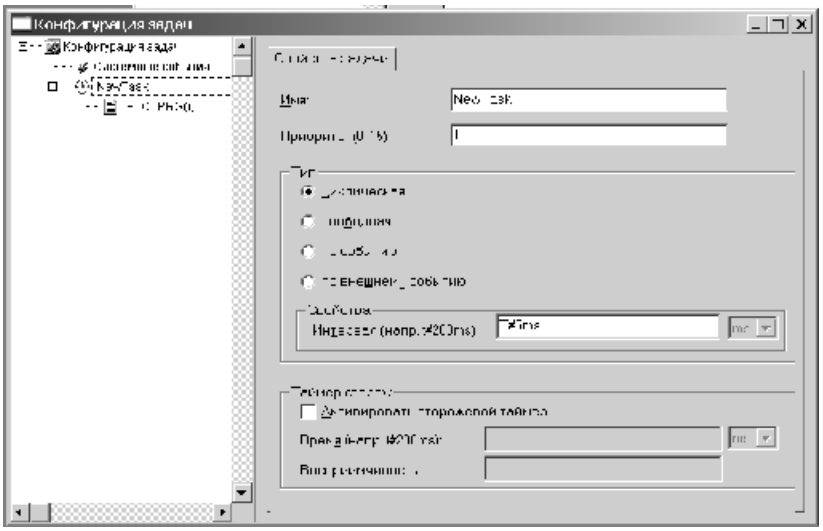

Рис 1. Настройка циклической задачи с вызовом программы PLC_PRG

При задании интервала циклической задачи нужно быть реалистом и не задавать физически недостижимых величин. Так, если используемая вами модель ПЛК имеет системный таймер, который 'тикает' раз в 10 мс, то установка интервала не кратного 10 мс не имеет смысла. Если в конфигурации ПЛК Овен вы задали время рабочего цикла 50мс, то чтобы вы не задали в конфигураторе задач, никакая задача не будет вызываться чаще.

При малых значениях интервала задачи весьма вероятно написать программу с настолько большим числом вычислений в одном цикле ее вызова, что она не будет 'помещаться' в заданный интервал. При этом интервал вызова программы будет нестабилен, что может быть нежелательно. Но как это проверить при отладке?

В ПЛК компании Овен для этой цели реализован встроенный механизм контроля времени рабочего цикла. В конфигурацию ПЛК необходимо добавить модуль статистики, который включает вход Cycle time. Здесь вы получите число, содержащее измеренное время цикла в сотнях миллисекунд.

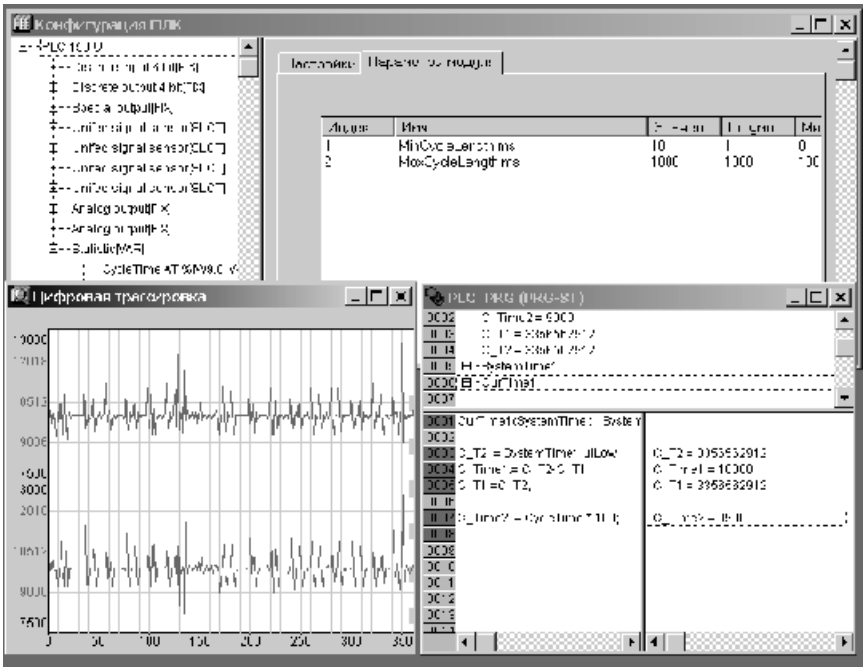

Рис 2. Измерение длительности цикла в ПЛК 150 Овен

Универсальным способом измерения коротких интервалов является использование точного таймера из системной библиотеки *SysLibTime.lib*. Библиотека содержит функциональный блок CurTime, который дает время системного таймера в микросекундах. Естественно, практически нельзя рассчитывать, что системный таймер любого ПЛК 'тикает' каждую микросекунду. Поэтому прежде чем полагаться на результаты этого способа измерения, уточните частоту его работы. Данная библиотека является не обязательной и может не

поддерживаться для определенных типов ПЛК. На *рисунке 2* показан результат прогона на ПЛК-150 Овен простой программы, которая измеряет интервал своего вызова в мкс, при установленном рабочем цикле 10мс. Для контроля приведены результаты измерений с использованием модуля статистики. С учетом дискретности таймера 100мкс мы получаем одинаковые результаты для обоих способов измерения.

Диалоговое окно 'Конфигурация задач' CoDeSys имеет свой встроенный диагностический инструмент. Для его работы необходимо включить в проект библиотеки SysLibTime и SysTaskInfo. Учтите что, они поддержаны не для всех типов ПЛК. На рисунке 3 показана информация о работе одной циклической задачи (5 мс), включающей программу, выполняющую сто тысяч операций с переменной типа REAL в CoDeSys SP RTE. На диаграмме видна заданная длительность цикла (светлая полоса) и реальное время вычислений. Оно занимает до 3.3 мс. Очевидно, что для данной задачи задавать интервал менее 4мс нет смысла.

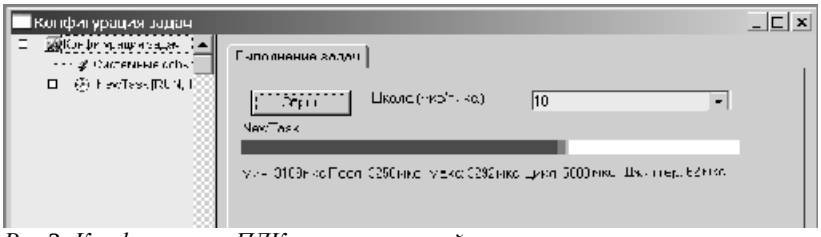

Рис 3. Конфигурация ПЛК в режиме онлайн.

Далеко не в каждом ПЛК удается напрямую измерить длительность выполнения программы. Часто она меньше или сравнима с дискретностью системного таймера. В восьмой статье данного цикла (Промышленные АСУ и контроллеры N1 2007) мы рассматривали пример измерения времени работы POU путем подсчета числа циклов, выполненных за 1 секунду. Вы можете воспользоваться данным методом, в случае если ваш ПЛК не имеет высокоточного системного таймера и не поддерживает библиотеку *SysLibTime*.

Итак, будем считать, что мы умеем делать в CoDeSys однозадачные проекты со стабильным интервалом цикла, знаем, как их отлаживать и проверять. Обладая такими навыками, можно переходить к освоению техники создания многозадачных приложений.

Корпоративная многозадачность

Допустим, нам нужно управлять тремя устройствами. Можно бы поставить три отдельных ПЛК, но устройства несложные и вполне можно попробовать обойтись одним контроллером. В таком случае, возникает логичная мысль не сваливать весь программный код в одну кучу, а разделить его на три программы.

Теперь представьте себе, что мы имеем обычный однозадачный ПЛК с классическим рабочим циклом: 1) чтение входов, 2) вызов прикладной программы, 3) запись выходов. Мы можем поступить очень просто: написать 3 программы и добавить их вызовы один за другим в главную прикладную программу:

PRG_A;
PRG_B;
PRG_C.

Вполне работоспособное решение. Но может оказаться, что общий цикл стал слишком медленным для PRG_A, хотя PRG_C можно бы, напротив, вызывать значительно реже. Тоже не проблема. Запрограммируем в главной программе некие счетчики, так чтобы каждая программа вызывалась не в каждом цикле, а только через определенное число циклов (например, 1 раз на 10 рабочих циклов). Изменяя параметры этих счетчиков, мы можем настраивать интервалы вызова наших программ. По сути дела мы реализовали три параллельно работающих задачи. Естественно, что в один момент времени работает только одна из них, но с позиции наблюдателя они работают параллельно.

Конечно, не очень удобно возиться со счетчиками. Это лишняя работа, лучше бы поручить ее системе исполнения. Собственно это и достигается с помощью конфигуратора задач CoDeSys.

Описанный выше механизм называется корпоративной многозадачностью либо многозадачностью без вытеснения. Корпоративность обозначает тот факт, что для нормальной работы всей системы, задачи должны сотрудничать между собой. Очевидно, что если одна из задач станет выполнять очень длинные вычисления, длиннее чем заданное время рабочего цикла, то она задержит выполнение других задач. Для корректной работы каждая задача не должна содержать длительных вычислений. Их необходимо разбивать на несколько циклов вызова задачи. Например, не использовать поиск в массиве с помощью итераций по циклу FOR, а обрабатывать каждый следующий элемент массива при очередном вызове задачи. В случае 'зацикливания' в одной задаче, останавливается все. Каждая задача должна аккуратно отработать свой цикл, прежде чем сможет получить управление другая задача. В этом главный минус корпоративной многозадачности. В каждой задаче нужно не забывать о других задачах и аккуратно избегать длительных вычислений в одном цикле вызова.

С другой стороны, в том, что каждая задача всегда дорабатывает свой цикл и не может быть прервана в произвольный момент, состоит и большой плюс. Мы можем абсолютно спокойно использовать результаты вычислений одной задачи в другой. Почему это так, станет понятно ниже при описании вытесняющей многозадачности. Отсюда следует, что 'корпоративная' многозадачность не означает 'плохая'. Часто она удобнее. В первую очередь тогда, когда задачи должны тесно взаимодействовать между собой. Без всяких ограничений в нескольких задачах можно использовать глобальные переменные любого типа. Все входы ПЛК опрашиваются перед вызовом каждой задачи, после каждой задачи происходит синхронное обновление всех выходов. Каждая задача может работать с произвольными входами/выходами, в том числе может контролировать выходы, изменяемые в другой задаче. Никакой опасности считать с них некие промежуточные, недостоверные значения нет. Практически мы пишем программы для нескольких корпоративных циклических задач точно так, как и в случае с одной задачей.

Если в программе допущена серьезная ошибка, приводящая к зацикливанию, то при корпоративной многозадачности контроллер полностью теряет работоспособность. Это равносильно фатальному сбою контроллера и его приходится перезапускать вручную. Преодолеть эту проблему в CoDeSys позволяют программные сторожевые таймеры. Алгоритм их работы предельно прост: если задача не возвращает управление дольше заданного времени, то таймер 'срабатывает' и происходит вызов специального обработчика системного события. По умолчанию этот обработчик выполняет перезапуск контроллера. Вы можете заменить его собственной реализацией, например, включить индикацию ошибки или инициализировать зависшую программу. Как писать обработчики системных событий, мы рассмотрим позднее.

Вытесняющая многозадачность

В случае вытесняющей многозадачности более приоритетная задача получает управление в положенное время, даже если менее приоритетные задачи еще не доработали до конца своего цикла. Задача с высоким приоритетом вытесняет задачу с меньшим приоритетом.

Основной плюс состоит в том, что задачи с низким приоритетом не влияют на задачи с высоким приоритетом. Каждая задача может делать сколь угодно длинные вычисления. Она автоматически будет прервана, при необходимости выполнить более приоритетную задачу и затем снова получит управление и продолжит работу с того места, где она была прервана. Таким образом, отпадает нужда думать об обеспечении заданного времени вычислений в рамках одного цикла вызова программы.

При вытесняющей многозадачности система исполнения CoDeSys плотно взаимодействует с операционной системой ПЛК. Каждая прикладная МЭК задача является отдельным потоком ОС.

Каждая задача имеет свой собственный рабочий цикл ПЛК. Опрос 'своих' входов, вызов 'своих' программ, запись 'своих' выходов происходит автономно, независимо от остальных задач. При компиляции проекта CoDeSys автоматически распределяет входы/выходы по задачам, в которых они задействованы и передает соответствующую конфигурацию драйверам аппаратуры.

Грубо можно считать, что при вытесняющей многозадачности мы имеем несколько ПЛК в одном корпусе. К сожалению, в CoDeSys V2.3 текст задач нельзя компилировать, перезагружать и отлаживать автономно, не нарушая работу других задач (это возможно в CoDeSys V3).

Проблемы в многозадачных проектах при вытесняющей многозадачности возможны при использовании общих аппаратных ресурсов и глобальных переменных. Представьте себе, что одна задача начала изменять некую глобальную переменную, например, структуру или строку. Пока изменение не полностью закончено, содержимое данной переменной не определено. Теперь представьте, что в этот момент другая задача получает управление и читает или, хуже того, записывает данную переменную. Результат непредсказуем. Ситуация может проявиться один раз после многих дней нормальной работы ПЛК. 'Поймать' ее отладчиком не реально. Остается только не допускать. Классическим приемом защитой от такой ситуации является использование семафоров. Одна задача захватывает некий ресурс для себя и взводит семафор, другие задачи ждут 'зеленый свет'. Семафоры в CoDeSys реализованы в библиотеке *SysLibSem.lib*. Работа с ними описана в файле SysLibSem_RU.pdf.

Очень важным является правильный выбор приоритетов задач в конфигураторе (см. рис. 1). Нулевое значение соответствует наивысшему приоритету. Свободная задача в проекте может быть только одна. Она занимает все свободное время процессора. Если присвоить ей высокий приоритет, то никакие менее приоритетные задачи никогда не получат управления. Всегда присваивайте свободной задаче низший приоритет.

Выше мы рассмотрели способы измерения длительности выполнения своих программ. Обратите внимание, что непредвиденные задержки выполнения способны вызывать и некоторые библиотечные функции. Допустим, функция осуществляет запись в файл или запрос данных от устройства по последовательному каналу связи. Очевидно, это требует определенного времени (до нескольких секунд). Есть два способа реализации функций. При **синхронной** реализации функция не возвращает управления вплоть до получения результата. Ее использование очень просто – вызываем функцию, получаем результат. Если такие функции сосредоточены в низкоприоритетной задаче, то их задержки не критичны, при вытесняющей многозадачности. Альтернативный вариант – это **асинхронная** реализация длительных операций. Одна функция начинает операцию и моментально возвращает управление. Некоторая отдельная

функция проверяет факт окончания операции. Ее нужно вызывать периодически. Задержек в функциях нет, но программа усложняется. Трудно сказать, что удобнее. Поэтому для работы с файлами в CoDeSys есть две библиотеки: SysLibFile с синхронными функциями и SysLibFileAsync с асинхронными функциями.

Альтернативой циклическим задачам является 'событийный' вызов задач. Вариант 'по событию' в конфигураторе задач *(см. Рис. 1)* позволяет вызвать данную задачу при изменении указанной логической переменной, например, дискретного входа. Такая задача также имеет приоритет и работает с вытеснением или корпоративно, как и циклические задачи.

В некоторых моделях ПЛК существуют устройства, способные генерировать так называемые 'внешние события'. Например, аппаратные таймеры, счетчики, компараторы и др. Вызов такой задачи может инициироваться аппаратным прерыванием.

Все отладочные средства CoDeSys доступны и в многозадачных проектах. Тонкость состоит в том, что для пошагового выполнения и контроля стека вызовов, CoDeSys нужно знать какую именно задачу мы отлаживаем. Не забудьте в конфигурации задач выбрать нужную и в контекстном меню дать команду 'Отлаживать эту задачу'.

Мы детально рассмотрели отличия корпоративной и вытесняющей многозадачности. Вытесняющая многозадачность реализована в ПЛК с CoDeSys, выполненных на базе 32-х разрядных процессоров с многозадачной ОС. Во всех остальных случаях в системе исполнения реализуется корпоративная многозадачность.

Как точно проверить, что в имеющемся у нас ПЛК реализована вытесняющая или корпоративная многозадачность? Давайте для практики напишем тестер качества реализации многозадачности. Если вы сможете самостоятельно придумать, как написать такую программу в CoDeSys, то это означает, что вы являетесь пользователем высшей квалификации и можете смело браться за любые задачи с многозадачностью. В противном случае, следуйте за изложением ниже.

Начнем с создания классической простейшей программы для ПЛК, инкрементирующей целую глобальную переменную при каждом вызове. Например, так:

```
g_udiQuickCounter := g_udiQuickCounter + 1;
```

Свяжем ее с циклической задачей названной QuickTask. Дадим ей высокий приоритет 5. Пусть она вызывается с интервалом в 2 мс.

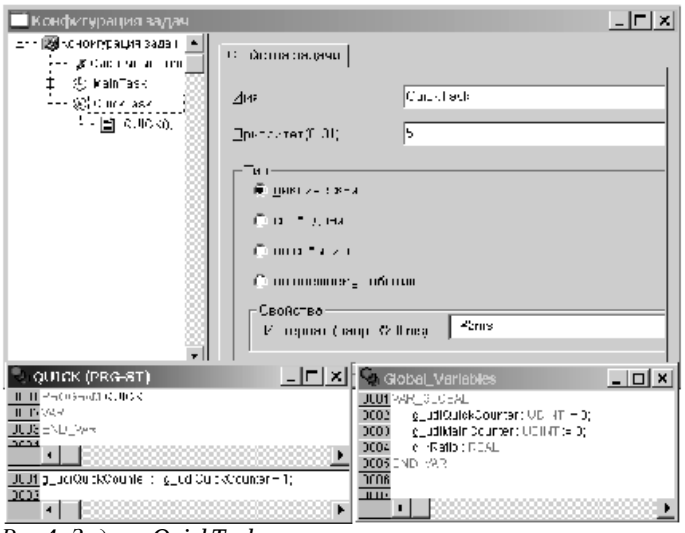

Рис 4. Задача QuickTask

Теперь делаем вторую 'вредную' программу. Она будет содержать аналогичный счетчик и делать некие сложные математические вычисления в цикле. Никакого смысла кроме траты процессорного времени в этих вычислениях нет:

```
g_udiMainCounter := g_udiMainCounter + 1;

FOR iActIndex := 0 TO iMaxIndex DO
    rArg := (2 * rPI * (iActIndex MOD 360)) / 360.0;
    rRes := 40 * SIN(rArg / (2 * rPI)) + 40;
END_FOR
```

Ее мы поместим в другую циклическую задачу с меньшим приоритетом 10 и интервалом 10мс. Вредность этой программы заключена в том, что она будет тратить на свои бессмысленные вычисления 8мс, оставляя

'окно' 2мс. При корпоративной многозадачности в это окно будет вписываться ровно один вызов первой задачи. Мы увидим, что оба счетчика меняются одинакового, несмотря на заданные разные интервалы задач. Соотношение счетчиков должно быть один к одному. При качественной вытесняющей многозадачности, более приоритетная задача будет вызываться четко в пять раз чаще. Соотношение счетчиков должно быть пять к одному. Отклонения от этих идеальных величин возможны, если выполнение системных сервисов (например, обслуживание сети) вызовет нарушение точности выдержки интервалов прикладных задач.

Подстраивать время вычислений нашей 'вредной' программы мы можем, изменяя число итераций цикла iMaxIndex. Конечно, его можно просто подобрать вручную под быстродействие испытуемого ПЛК, дабы вычисления занимали 8мс. Но это не технично. Добавим в программу регулятор, который будет автоматически поддерживать заданную длительность цикла вычислений:

```
TickBase(SystemTime:=SystemTime);
t1 := SystemTime.ulLow;

(* тут стоит цикл вычислений FOR, см выше*)

TickBase(SystemTime:=SystemTime);
t2 := SystemTime.ulLow;
t3 := t2 - t1;

IF t3 < 7950 THEN
    iMaxIndex := iMaxIndex + 100;
END_IF

IF t3 > 8050 AND iMaxIndex > 1000 THEN
    iMaxIndex := iMaxIndex - 100;
END_IF
```

Здесь TickBase – это системный таймер CurTime из библиотеки *SysLibTime*. При объявлении переменная iMaxIndex получает начальное значение 20000. В данном фрагменте она регулируется так, чтобы интервал задачи составлял 8000мкс +-50.

Осталось вычислить соотношение счетчиков:

```
g_rRatio := UDINT_TO_REAL(g_udiQuickCounter) / UDINT_TO_REAL(g_udiMainCounter);
```

Полная реализация этой программы и конфигурация задачи показана на *рис. 5*.

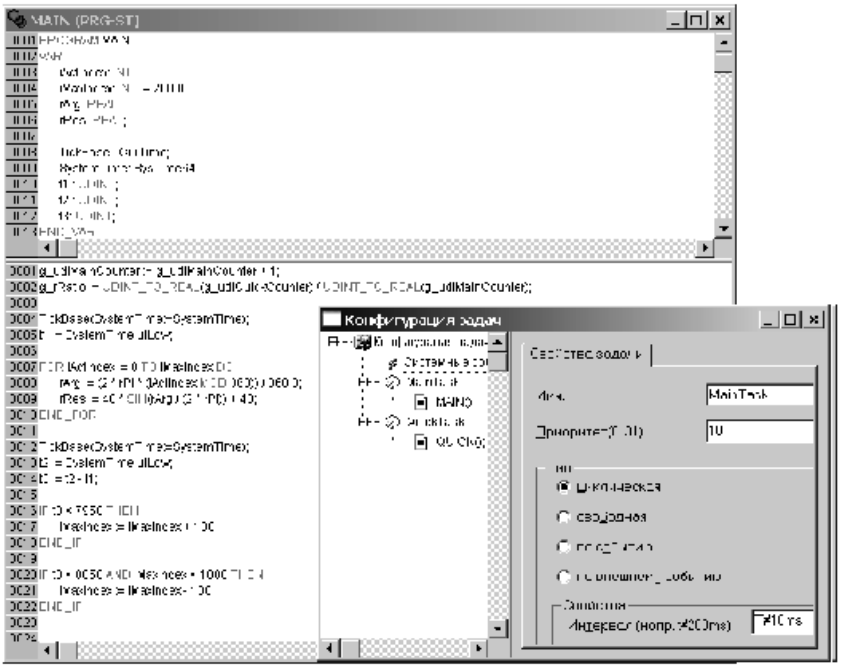

Рис 5.Реализация задачи MainTask

Результаты запуска тестовой программы таковы:

CoDeSys SP RTE: соотношение 4.99. Ничего удивительного, так и должно быть в качественном контроллере с вытесняющей многозадачностью.

Овен ПЛК-150: соотношение 1.00. Проследив трассировкой значение в течении нескольких минут, мы видим стабильную единицу практически без джиттера. Можно сделать вывод, что в данном ПЛК реализована корпоративная многозадачность, причем работает она идеально.

Итак, для использованных нами ПЛК множество задач не стало проблемой. А что будет, если один ПЛК одновременно начнут программировать множество людей? В CoDeSys это возможно, но тут пригодится один нехитрый прием.

Продолжение следует.