2. Конвейерную схему организации работы станций, примененную для построения диаграммы и анализа времени выполнения прикладных функций, следует рассматривать как граничную допустимую возможность организации выполнения графа потока данных с соблюдением заданного времени обновления выходов прикладных функций. Анализ диаграммы позволяет разработчику проекта относительно каждой прикладных функций выявить наиболее значительные задержки времени ее выполнения и принять решения по снижению таких задержек.
3. Исследования конвейерной схемы организации выполнения графа потока данных с целью сокращения запаздывания выполнения прикладных функций позволит в последующем в большей мере формализовать связи между величиной запаздывания и параметрами, определяющими архитектуру вычислительной системы, распределение модулей и данных по станциям, условия динамики работы графа потока данных, условия распараллеливания, селекции состояний, приоритетности.

СПИСОК ЛИТЕРАТУРЫ

1. Погребной В.К. Визуальный уровень представления алгоритмов функционирования распределенных систем реального времени на языке структурного моделирования // Известия Томского политехнического университета. – 2009. – Т. 314. – № 5. – С. 140–146.
2. Погребной А.В. Определение числа и топологии размещения станций многопроцессорной вычислительной системы // Известия Томского политехнического университета. – 2006. – Т. 309. – № 7. – С. 160–164.
3. Погребной В.К., Погребной А.В., Погребной Д.В. Отображение условий динамики функционирования объекта управления на модель системы реального времени // Известия Томского политехнического университета. – 2009. – Т. 315. – № 5. – С. 18–22.
4. Шенброт И.М., Алиев В.М. Проектирование вычислительных систем распределенных АСУ ТП. – М.: Энергоатомиздат, 1989. – 88 с.
5. Погребной А.В., Погребной Д.В. Проектирование структуры локальной сети для распределенной вычислительной системы реального времени // Известия Томского политехнического университета. – 2007. – Т. 311. – № 5. – С. 97–101.

Поступила 05.11.2009 г.

УДК 004.031.6

РАЗРАБОТКА ВСТРОЕННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ КОНТРОЛЛЕРОВ, ИСПОЛЬЗУЕМЫХ В ОБЛАСТИ ПРОМЫШЛЕННОЙ АВТОМАТИЗАЦИИ

К.С. Щербаков, С.А. Щербаков, В.А. Кочегуров

Томский политехнический университет
E-mail: konstantin.sherbakov@elesy.ru

Представлен метод разработки встроенного программного обеспечения для программируемых логических контроллеров. Метод основан на преобразовании критических участков кода в критические секции, с возможностью целостного исполнения кода процессов.

Ключевые слова:
Разграничение доступа, критический участок и критическая секция, целостное исполнение кода программ.

Key words:
Access isolation, critical region and critical section, entire performance of program code.

Для обеспечения управления технологическими процессами в промышленности и другими сложными технологическими объектами используются, так называемые, программируемые логические контроллеры (ПЛК). Основная работа ПЛК сводится к сбору сигналов от датчиков, их обработке прикладной программой пользователя и выдаче управляющих сигналов для исполнительных устройств [1]. Функциональность ПЛК зависит не только от аппаратного, но и от программного обеспечения (ПО), которое все чаще называют встроенным программным обеспечением (ВПО или Embedded Software). В настоящее время именно к ВПО, как к неотъемлемой части любого ПЛК, предъявляются новые и более жесткие требования, связанные с устойчивостью, многофункциональностью и надежностью работы ПЛК в целом.

Разработка ВПО зачастую затруднена следующими ограничениями, обусловленными особенностями архитектуры микроконтроллеров, на базе которых разрабатывается сам ПЛК и спецификой задач, решаемых ПЛК, а именно:

- Аппаратно-программным модулям, которые входят в состав ПЛК и обеспечивают функции по сбору сигналов от датчиков и выдаче управляющих сигналов для исполнительных устройств, необходимо обрабатывать источники прерываний, количество которых более пяти.
- Код критических участков (код, исполняемый при закрытых прерываниях) выполняется достаточно продолжительное время и зачастую в сумме превосходит время реакции модуля на внешнее событие (получение сигнала от датчика, запуск исполнительного механизма и т. п.).
- Прерывания имеют одинаковый приоритет исполнения, то есть одно прерывание не вытесняет другое прерывание.
- Существует необходимость в декомпозиции задачи, с целью параллельной реализации программного кода группой программистов-разработчиков.

В тоже время, создаваемое ВПО должно удовлетворять следующим требованиям:

- Все источники прерываний должны быть обслужены, а возможность потери прерывания, на которое всегда возложена функция об уведомлении о наступившем событии, должна быть минимизирована.
- Критические участки кода программ должны выполняться целостно, без запрещения возникновения прерываний.
- Время выполнения системных функций ВПО должно быть точно подсчитано, для определения реакции системы на внешнее событие.

Для того, чтобы разрабатываемое ВПО удовлетворяло вышеописанным требованиям, сотрудники компании «ЭлеСи» (г. Томск) разработали специализированный программный продукт и назвали его Микроядро1. Микроядро1 – это инструмент, который помогает разработчику декомпозировать задачу на этапе разработки ВПО. Механизмы, реализованные в Микроядре1, позволяют выполнять код критических участков целостно, без запрещения возникновения прерываний другим источникам [2].

Для определения принципов работы Микроядра1 введем основные понятия, которые используются в статье:

Критический участок – небольшой участок кода процесса, при выполнении которого прерывания всем источникам запрещены.

Критическая секция – продолжительный участок кода, исполняемый целостно (выполняется без прерываний со стороны других процессов или других критических секций), при открытых прерываниях всем источникам.

Критический ресурс – ресурс, который в каждый момент времени используется не более чем одним процессом [3].

Фоновая задача (ФЗ) – часть программного кода, выполнение которого организованно как бесконечный процесс. Фоновая задача состоит из некоторого количества вызовов пользовательских подпрограмм. Порядок вызова пользовательских подпрограмм может быть произвольным, а фоновая задача имеет самый низкий приоритет исполнения.

Прерывание – сигнал, по которому процессор «узнает» о совершении асинхронного события. При этом исполнение текущей последовательности команд приостанавливается, а вместо нее начинает исполняться другая последовательность, соответствующая данному прерыванию.

Обработчик прерываний (ОП) – программная секция, которой передается управление в случае прерывания процессора.

Отложенное прерывание (ОТП) – асинхронно исполняемая часть программы обработки прерывания. Данный участок программного кода выполняется целостно, прерывания всем источникам разрешены. ОТП выполняются целостно (не вытесняют друг друга).

Приоритетный процесс (ПП) – процесс выполнения кода пользовательских задач, которые выполняется целостно в соответствии с приоритетом, прерывания всем источникам разрешены.

Директива – это код программы, оформленный с помощью функций Микроядра1, который выполняется целостно по отношению к ОТП, но является более приоритетным, чем ПП и Фоновая задача.

Очередь – последовательный список, в котором все дополнения вносятся в один конец, а доступ для выборки осуществляется к другому концу списка [3].

Микроядро1 состоит из четырех функций, написанных на платформенно-ориентированном языке (Ассемблер). Объем кода Микроядра1 не превышает 900 строк. Время выполнения функций Микроядра1 может быть точно подсчитано, в зависимости от выбранной микропроцессорной платформы, на которую будет портированно Микроядро1. Экспериментальным путем подсчитано, что время выполнения функций для микропроцессорных архитектур с производительностью равной 100 MIPS (Million Instruction Per Second) занимает десятые доли микросекунды.

С помощью функций Микроядра1 _delay_isr обработчик прерывания оформляется по определенному правилу, которое заключается в том, что код обработчика прерывания делится на две части, рис. 1.

Как следует из рис. 1, код обработчика прерывания состоит из двух частей. Первая часть – 1 выполняется немедленно после возникновения прерывания. Вторая часть – 2, которая соответствует ОТП, будет поставлена в очередь и выполнена позже. Каждому запланированному ОТП будет соответствовать элемент очереди – дескриптор ОТП.

Если обработчик прерывания состоит из небольшого числа ассемблерных инструкций, то он может быть выполнен целиком. В этом случае нет необходимости создавать ОТП.

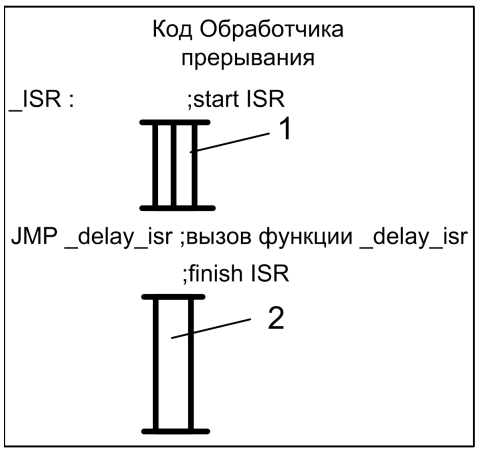

Рис. 1. *Схема обработки источника прерываний: 1) код выполнится сразу после возникновения прерывания, 2) код ОТП*

Главная особенность такой декомпозиции кода обработчика прерывания заключается в том, что разработчик ПО самостоятельно решает какую часть кода необходимо выполнять как критический участок, а оставшаяся часть кода будет выполнена как критическая секция. Программный код, оформленный в виде ОТП, выполняется целостно (без вытеснения) со стороны всех ОТП, запланированных от других источников прерываний.

Директивы предназначены для выполнения критического участка кода программы, как кода критической секции. Пользователь может написать свои директивы, используя функцию Микроядра1 – _dir. Директива может быть вызвана из Фоновой задачи, из ОТП или из ПП. Директива выполняется целостно и при открытых прерываниях, следовательно, отложенные ОТП во время исполнения директивы будут поставлены в очередь.

Код программы, осуществляющей доступ к критическому ресурсу, разработчику необходимо оформить в виде Директивы. Директивы позволяют получить разграничения доступа при обращении к критическому ресурсу(совместно используемая память и т. п.). Директива позволяет заменить критические участки, которые выполняются на уровне ОТП, и на уровне Фоновой задачи или на уровне ОТП и на уровне ПП, критическими секциями, которые не сдерживают выполнение кода обработчика прерывания.

Если директива вызвана в ОТП, то после ее выполнения будет продолжено выполнение ОТП. После выполнения директивы, вызванной из Фоновой задачи или из ПП, управление будет передано в функцию Микроядра1 – _disp, которая последовательно запустит все запланированные ОТП из очереди, рис. 2.

Фоновая задача (ФЗ) состоит из последовательного вызова пользовательских функций. Существует необходимость выполнять код некоторых функций приоритетно по отношению к другим функциям. Например, вывод состояния индикации устройства менее приоритетно, чем обслуживание WDT (WatchDog Timer). В связи с этим, разработчики Микроядра1 реализовали системную директиву – _sys_dir, которая выполняет постановку ПП в очередь на исполнение, а очередь формируется в соответствии с приоритетом ПП. То есть, ПП с наибольшим приоритетом будет выполнен в первую очередь, но целостно по отношению к другим ПП.

При выполнении ПП может произойти прерывание. Соответствующий прерыванию ОТП будет выполнен приоритетно по отношению к любому ПП, рис. 3.

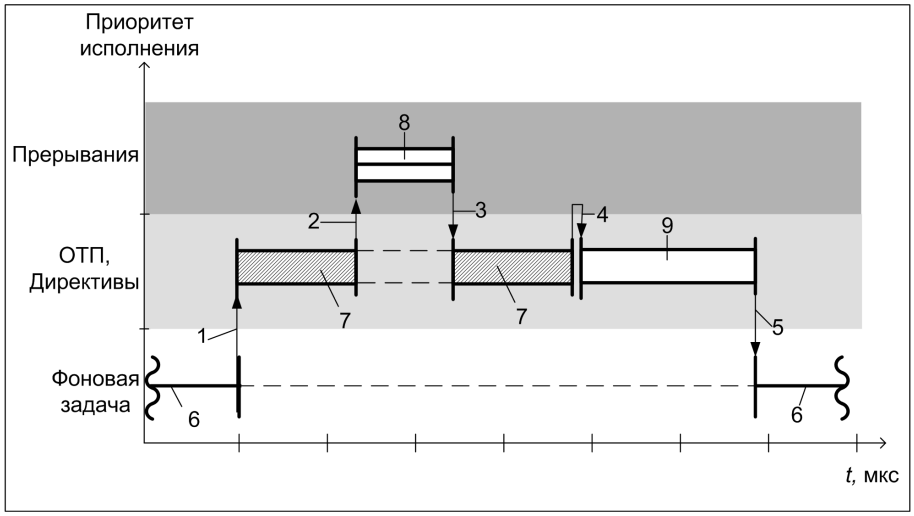

Рис. 2. *Схема выполнения кода Директив, ОП и ОТП: 1) вызов Директивы, 2) переход на выполнение ОП, 3) переход на выполнение Директивы (по завершению выполнения ОП), 4) переход на выполнение ОТП (по завершению выполнения Директивы), 5)переход на выполнение Фоновой задачи, 6) код Фоновой задачи, 7) код Директивы, 8) код ОП, 9) код ОТП*

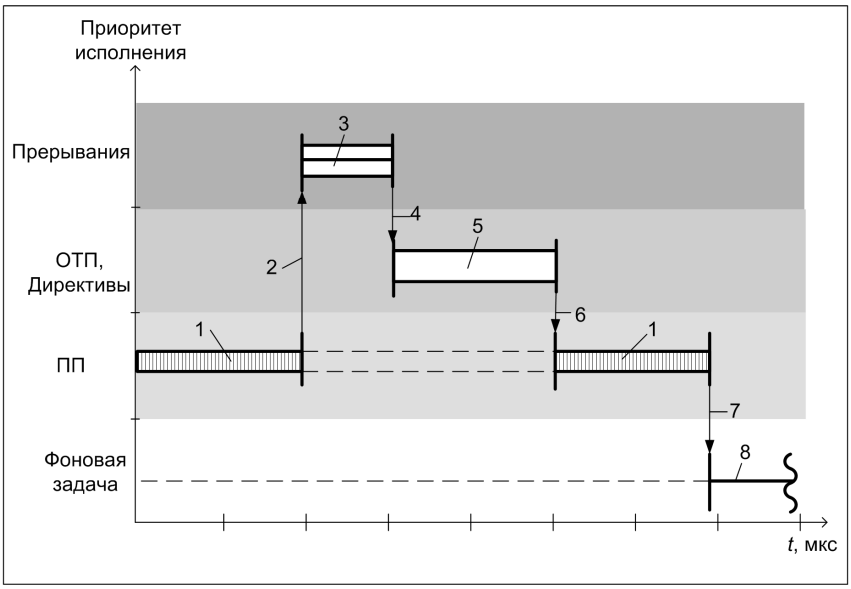

Рис. 3. *Схема выполнения ПП: 1) код ПП; 2) переход на выполнение ОП; 3) код ОП; 4) переход на выполнение ОТП; 5) код ОТП; 6) переход на выполнение ПП (по завершению ОТП); 7) переход на выполнение Фоновой задачи; 8) код Фоновой задачи*

Как показано на рис. 3, код ОП обладает наивысшим приоритетом и прерывает выполнение ОТП, ПП и Фоновой задачи.

Примером использования Микроядра1 может послужить реализация задачи приема данных, их обработки и передачи данных, которая очень часто решается разработчиками ВПО. Данную задачу разобьем на пять последовательных этапов.

Этап 1:

При приеме данных возникает прерывание, которому соответствует код ОП, рис. 4.

По завершению выполнения ОП данные находятся в буфере приема. Процесс копирования данных в буфер обработки может занять продолжительное время, поэтому выполним код копирования отложено на соответствующем ОТП.

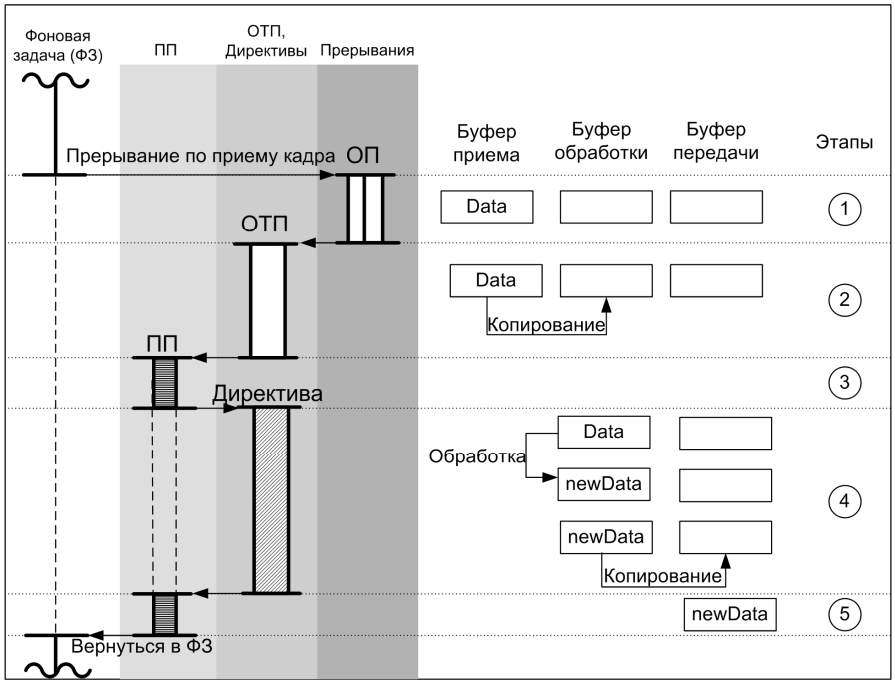

Рис. 4. *Реализация задачи приема и обработки данных с помощью Микроядра1*

Этап 2:

При выполнении ОТП принятые данные будут скопированы в буфер обработки. По завершению этапа 2 будет запланирован ПП обработки принятых данных.

Этап 3:

Состоит из вызова Директивы.

Этап 4:

В теле Директивы принятые данные будут обработаны и скопированы в буфер передачи. Если во время выполнения Директивы произойдет прерывание, то ОП будут выполнен, а соответствующий ОТП встанет в очередь и выполнится по завершению выполнения кода Директивы.

Этап 5:

Завершающее действие на уровне ПП – это передача данных.

Достоинством реализации поставленной задачи служит то, что действия по приему данных и по их обработке разграничены, и выполнятся целостно на уровне ОТП и Директивы. При выполнении программного кода, начиная с 2-го и по 5-й этап – прерывания всем источникам разрешены, а значит время задержки на обработку источников прерываний приведены к минимуму. Код обработки кадра выполнен на уровне ПП, приоритетно по отношению к Фоновой задаче, где функции выполняются последовательно.

Недостатком данной реализации является наличие накладных расходов при выполнении функций Микроядра1.

Функции Микроядра1, которые определяют логику выполнения ОП, ОТП, Директив, ПП и Фоновой задачи, естественным образом используют часть системных ресурсов (память, время работы микропроцессора и т. д.). С целью минимизации накладных расходов системных функций разработчики Микроядра1 использовали язык низкого уровня (Ассемблер).

Выводы

Представлен метод разработки встроенного программного обеспечения для программируемых логических контроллеров под названием Микроядро1, основанный на преобразовании критических участков кода в критические секции, с возможностью целостного исполнения кода процессов.

Использование Микроядра1 позволяет минимизировать время выполнения критических участков кода и ускорить реакцию системы на внешнее событие. Директива служит цели разграничения доступа к общему ресурсу, который используется несколькими процессами без использования семафоров. Разработка встроенного программного обеспечения с использованием принципов Микроядра1 структурирует программный код, упрощает процесс отладки и уменьшает сроки разработки встроенного программного обеспечения при минимальных накладных расходах. Микроядро1 может быть использовано при создании встроенного программного обеспечения модулей программируемых логических контроллеров, решающих такие задачи, как измерение и выдача аналоговых и дискретных сигналов, а также задачи коммуникации для разного рода протоколов передачи данных.

СПИСОК ЛИТЕРАТУРЫ

1. Нестеренко П.Г. Программируемые логические контроллеры России: Реальность и перспективы // Журнал интеллектуальных технологий Айтэч. – 2008. – Т. 1. – № 11. – С. 6–12.
2. Щербаков К.С. Разработка и реализация системного программного обеспечения для однокристальных микроконтроллеров // Молодежь и современные информационные технологии: Матер V Междунар. научно-практ. конф. молодых ученых. – г. Томск, 27 февраля–1 марта 2007. – Томск, 2007. – С. 462–464.
3. Толковый словарь по вычислительным системам / Под ред. В. Иллингуорта и др.: Пер. с англ. А.К. Белицкого и др.; Под ред. Е.К. Масловского. – М.: Машиностроение, 1990. – Т. 1. – С. 110–387.

Поступила 18.05.2009 г.