Межпроцессорное взаимодействие (IPC) в многоядерных микроконтроллерах.
Часть 5. Использование аппаратных блокировок модуля SEMA4

В предыдущих статьях были подробно рассмотрены варианты реализации межпроцессорного взаимодействия в многоядерных гетерогенных микроконтроллерах и микропроцессорах, одним из ядер которых является ядро Cortex-M3/M4. В предлагаемой публикации на примере программирования процессоров MPC56xx и MPC57xx семейства Qorivva компании FreeScale (в настоящее время NXP) рассматриваются вопросы практического использования аппаратных блокировок, а также производится сравнение аппаратных блокировок с программными блокировками (семафорами). Изложенный материал остается справедливым и для других новейших семейств многоядерных микросхем компании FreeScale, в состав которых входит аппаратный модуль блокировок (hardware semaphores module) SEMA4, таких как i.MX6, i.MX7 и i.MX8.

Андрей САМОДЕЛОВ

Введение

Для понимания разницы между программной и аппаратной реализацией элементов блокировки и особенностей их использования кратко опишем основные свойства программных элементов блокировки — семафоров.

Семафор — это переменный или абстрактный тип данных, обеспечивающий простую, но полезную абстракцию для управления доступом нескольких процессов к общему ресурсу в среде параллельного программирования или многопользовательском окружении.

Семафоры являются преобладающим методом контроля доступа в параллельном программировании и многопоточных средах программирования с момента их изобретения в 1965 году Эдсгером Дейкстрой (Edsger Dijkstra), голландским ученым в области компьютерных наук. Хотя концепция семафоров была еще более расширена из первоначального принципа на многие вариации, все они служат одной и той же цели — на равных правах предоставить процессам доступ к общим ресурсам.

Вернемся к аппаратным элементам блокировки. Практический подход к аппаратным аналогам программных семафоров (hardware semaphores) предоставляет простые процедуры управления (semaphore routines), которые блокируют, разблокируют и сбрасывают аппаратные элементы блокировки (semaphore gates), аналогичные флагам программных семафоров. Эти процедуры управления являются неблокирующими, а повторно используемый код функций подходит для многоядерных приложений. Далее для каждой процедуры будет представлен пример кода на языке С, за которым следует подробное объяснение ее функциональности.

После обсуждения каждой из подпрограмм управления (semaphore routines) элементами блокировки (gate) они будут использованы для создания нескольких классических примеров семафоров (semaphore examples):

- простейший сигнальный семафор (signaling semaphore);
- блокирующий семафор (blocking semaphore);
- семафор для встречных процессов (rendezvous semaphore).

Рассмотренные примеры создавались для микропроцессора MPC5643L, но они могут быть использованы для любых многоядерных микросхем MPC56xx и MPC57xx семейства Qorivva, а также других микропроцессоров с аналогичными модулями аппаратных блокировок (hardware semaphore module).

Цель дальнейшего изложения — обеспечить необходимое понимание, достаточное для грамотного использования аппаратного модуля блокировок (hardware semaphore module) многоядерных микросхем MPC56xx и MPC57xx семейства Qorivva. Оно не претендует на то, чтобы являться изложением концепции семафоров в целом.

Предварительные требования

Для понимания приведенных далее программных примеров необходимо знакомство с языком ассемблера для PowerPC, а также владение программированием на языке С.

Для написания и отладки приведенных примеров программ была использована среда разработки CodeWarrior [2].

В качестве аппаратной платформы использованы:

- кросс-плата XPC56xx с мини-модулем XPC5643L144QFP (EVB, целевая система);
- внешний источник питания с напряжением 12 В постоянного тока для питания оценочной платы (EVB);
- персональный компьютер с ОС Windows (базовая система).

Процедуры управления элементами блокировки

Все представленные далее процедуры управления элементами блокировки (semaphore routines) являются платформенно-зависимыми. Они будут работать только на многоядерных микросхемах MPC56xx и MPC57xx семейства Qorivva с аппаратными модулями блокировок (hardware semaphore module), как представлено в таблице 1.

Перейдем к описанию конкретных процедур работы с модулем управления блокировками.

Таблица 1. Многоядерные микросхемы с аппаратными модулями блокировок

Микросхема	Тип модуля блокировок (Semaphore module)	Замечания
MPC5643L (в режиме Decoupled Parallel mode)	SEMA4	Первое поколение аппаратного модуля блокировок (hardware semaphore module)
MPC5643L (в режиме Lock Step mode)	Отключено в режиме Lock Step mode	–
MPC5676R	SEMA4	Первое поколение аппаратного модуля блокировок (hardware semaphore module)
MPC57xxM (многоядерные)	SEMA42	Второе поколение аппаратного модуля блокировок (hardware semaphore module)

Примечание. При описании модуля аппаратных блокировок (hardware semaphore module) компании FreeScale используется термин «элемент блокировки» (semaphore gate), который соответствует термину «флаг семафора» (semaphore flag) при программной реализации. Термин «заблокированный элемент» (locked gate) аналогичен термину «установленный флаг» (set flag), а термин «разблокированный элемент» (unlocked gate) и «сброшенный элемент» (resetted gate) аналогичны термину «очищенный флаг» (cleared flag).

Процедура получения идентификатора ядра процессора, Get_Core_ID

Для аппаратного модуля управления блокировками (hardware semaphore module) необходимо, чтобы для всех ядер, претендующих на изменение состояния элемента блокировки (semaphore gate), в качестве одного из необходимых параметров блокировки присутствовал оттранслированный идентификатор ядра (core ID). Рассматриваемая подпрограмма читает идентификатор ядра (core Process ID, PID) и преобразует его в эквивалентный идентификатор (equivalent ID), который сможет принять аппаратный модуль управления блокировками (hardware semaphore module).

Данную подпрограмму можно вызывать единственный раз за время исполнения приложения, кроме того, она может быть выполнена из подпрограммы Lock Gate, или пользователь просто предоставит (вручную, в коде) необходимый идентификатор ядра (core ID) в подпрограмму Lock_Gate. В дальнейших примерах, для простоты и безопасности, функция Get_Core_ID вызывается внутри подпрограммы Lock_Gate. Этот метод также гарантирует, что для изменения состояния элемента блокировки (gate) в аппаратный модуль управления блокировками (hardware semaphore module) будет отправлен правильный идентификатор ядра (core ID).

```
//-----------------------------------------------------------
// Get_Core_ID
//-----------------------------------------------------------
1: uint8_t Get_Core_ID (void)
2: {
3:       registeruint32_t temp;
4:
5:       asm ("mfspr temp, 286");
6:
7:       switch (temp)
8:       {
9:              case 0: return Core0_ID;
10:             case 1: return Core1_ID;
11:      }
12: }
```

Листинг 1. Подпрограмма Get_Core_ID

В листинге 1 приведен пример исходного кода на языке C для подпрограммы Get_Core_ID. Эта подпрограмма считывает core ID и возвращает в вызывающую процедуру оттранслированный идентификатор core ID. В таблице 2 приведено описание кода, представленного в листинге 1.

Приведенная в листинге 1 подпрограмма преобразует идентификаторы (ID) для двухъядерной системы. Чтобы изменить приведенную выше подпрограмму для поддержки большего количества ядер, достаточно просто увеличить количество веток оператора выбора case в соответствии с реальным количеством ядер и возвращать нужным образом преобразованный идентификатор core IDs.

Таблица 2. Описание кода подпрограммы Get_Core_ID, представленного в листинге 1

№ строки	Описание
1	Подпрограмма Get_Core_ID будет возвращать оттранслированный code ID в вызывающую процедуру.
5	Метод вызова ассемблерной инструкции (assembly code) из подпрограммы на языке C для чтения регистра специального назначения (SPR) с номером 286 (PID). Этот регистр (SPR) содержит PID. (Для получения информации о правильном номере регистра SPR для конкретного процессора необходимо обратиться к соответствующей документации.)

7–10	Транслирует core ID и возвращает вызывающей процедуре. (Подробности о правильном значении идентификатора ядра процессора (processor ID, PID) можно найти в описании SEMA4_GATEn в разделе Semaphore документа Reference Manual на соответствующую микросхему.) Чтобы получить из идентификатора ядра (PID) идентификатор (ID), понятный модулю управления блокировками (semaphore module), необходимо выполнить следующее преобразование:
	Преобразованный_CoerID = { PID+1; 0+1 = 1 // если_PID = 0; 1+1 = 2 // если_PID = 1 }
	Константы:
	Core0_ID = 1 // — преобразованный core ID для ядра 0, используемый для передачи в модуль управления блокировками (semaphore module).
	Core1_ID = 2 // — преобразованный core ID для ядра 1, используемый для передачи в модуль управления блокировками (semaphore module).

Процедура получения значения состояния элемента блокировки, Get_Gate_Status

Подпрограмма считывает состояние конечного автомата (state machine) элемента блокировки (gate) и возвращает состояние вызывающей процедуре. Из состояния элемента блокировки (gate) пользователь может определить, какое из ядер его заблокировало или же элемент блокировки доступен для всех ядер (gate). Разблокированный (unlocked) элемент блокировки (gate) будет возвращать нулевое значение состояния, в то время как заблокированный (locked) элемент блокировки (gate) будет возвращать значение, связанное с номером заблокировавшего его ядра.

```
//-----------------------------------------------------------
// Get_Gate_Status
//-----------------------------------------------------------
1: uint8_t Get_Gate_Status (uint8_t gate_no)
2: {
3:       return SEMA4.GATE [gate_no].R;
4: }
```

Листинг 2. Подпрограмма Get_Gate_Status

Таблица 3. Описание кода подпрограммы Get_Gate_Status, представленного в листинге 2

№ строки	Описание
1	Get_Gate_Status считывает данные из запрашиваемого элемента блокировки (gate) и возвращает значение его состояния (gate status) вызывающей процедуре.
3	Считывает состояние конечного автомата элемента блокировки (gate state machine) и возвращает значение его состояния (gate status) вызывающей процедуре.

В листинге 2 приведен исходный код подпрограммы Get_Gate_Status на языке C. Эта подпрограмма считывает состояние конечного автомата (state machine) элемента блокировки (gate) и возвращает состояние вызывающей процедуре. В таблице 3 приведено описание кода, представленного в листинге 2.

Процедура блокировки элемента блокировки, Lock_Gate

Для получения информации о количестве поддерживаемых элементов блокировки (gates) можно обратиться к разделу Semaphore документа Reference Manual, относящегося к используемой микросхеме. MPC5643L в режиме Decoupled Parallel mode (DPM) содержит 16 элементов блокировки.

Каждый из элементов блокировки (gate) может быть заблокирован любым ядром процессора (core), а разблокирован только

Таблица 4. Описание кода подпрограммы Lock_Gate, представленного в листинге 3

№ строки	Описание
1	Подпрограмма Lock_Gate пытается блокировать указанный вызывающей процедурой элемент блокировки (gate) и возвращает его состояние (status gate) после завершения попытки блокировки.
3	Вызов подпрограммы Get_Core_ID для получения идентификатора ядра.
7	Попытка заблокировать запрошенный элемент блокировки (gate) с использованием оттранслированного идентификатора ядра core ID.
9	Вызов подпрограммы Get_Gate_Status для получения значения состояния элемента блокировки.

```
//-------------------------------------------------------------
// Lock_Gate
//-------------------------------------------------------------
1: uint8_t Lock_Gate (uint8_t gate_no)
2: {
3:      uint8_t Core_ID;
4:
5:      Core_ID = Get_Core_ID ();
6:
7:      SEMA4.GATE [gate_no].R = Core_ID;
8:
9:      return Get_Gate_Status (gate_no);
10: }
```

Листинг 3. Подпрограмма Lock_Gate

тем ядром, которое его ранее заблокировало. Вызывающая эту подпрограмму процедура для блокировки элемента блокировки (gate) должна передать в качестве параметра его правильный номер (gate number); неверные номера элементов блокировки (gate numbers) будут проигнорированы. Кроме того, вызывающая процедура должна разобрать (parse) возвращенное значение состояния элемента блокировки (gate status), чтобы определить, успешно ли была выполнена операция. В случае успешного завершения операции будет возвращено значение состояния (status), равное оттранслированному идентификатору ядра (core ID), связанное с заблокировавшим элемент блокировки (gate) процессором.

В листинге 3 приведен пример исходного кода на языке C для подпрограммы Lock_Gate. Эта подпрограмма пытается блокировать элемент блокировки (gate) и возвращает значение состояния элемента блокировки (gate) вызывающей процедуре. В таблице 4 приведено описание кода, представленного в листинге 3.

Подпрограмма разблокировки элемента блокировки, Unlock_Gate

Элемент блокировки (semaphore gate) может быть разблокирован только тем ядром (core), которое его перед этим заблокировало. Однако один или несколько заблокированных элементов блокировки могут быть выведены из заблокированного состояния (locked state) также с помощью операции сброса (reset),

1 Далее для обозначения флажка аппаратного семафора будет продолжено использование термина «элемент блокировки», в то время как при упоминании программных объектов будут употреблены термины «семафор» (semaphore) и «флажок (семафора)» (semaphore gate).

Таблица 5. Описание исходного кода подпрограммы Unlock_Gate, представленного в листинге 4

№ строки	Описание
1	Подпрограмма Unlock_Gate предпринимает попытку разблокировать запрошенный вызывающей процедурой элемент блокировки (gate) и возвращает значение состояния (gate status) этого элемента блокировки (gate).
3	Попытка разблокировать элемент блокировки (gate). Элемент блокировки (gate) может быть разблокирован только тем ядром (core), которое его заблокировало. Константы: UNLOCK = 0.
5	Вызов подпрограммы получения состояния элемента блокировки Get_Gate_Status.

```
//-------------------------------------------------------------
// Unlock_Gate
//-------------------------------------------------------------
1: uint8_t Unlock_Gate (uint8_t gate_no)
2: {
3:      SEMA4.GATE [gate_no].R = UNLOCK;
4:
5:      return Get_Gate_Status (gate_no);
6: }
```

Листинг 4. Подпрограмма Unlock_Gate

предоставляемой аппаратным модулем управления блокировками (hardware semaphore module). Подробно сброс элемента блокировки описан далее в разделе «Подпрограмма сброса элемента блокировки (Reset Semaphore Gate), Reset_Gate». Вызвавшая подпрограмму Unlock_Gate процедура должна также разобрать (parse) возвращенное ей значение состояния элемента блокировки (gate status), чтобы определить, была ли успешно завершена требуемая операция.

В листинге 4 приведен пример исходного кода на языке C для подпрограммы Unlock_Gate. Эта подпрограмма делает попытку разблокировать элемент блокировки (gate) и возвращает в вызывающую процедуру значение состояния элемента блокировки (gate status). В таблице 5 приведено описание кода, представленного в листинге 4.

Подпрограмма сброса элемента блокировки, Reset_Gate

Возможность сброса отдельного элемента блокировки (Reset semaphore gate) предоставляет любому ядру гибкий механизм сброса одного или нескольких элементов блокировки (gates) в состояние готовности (ready state), или разблокированное состояние. В частности, подпрограмма Reset_Gate будет выводить один или несколько заблокированных элементов блокировки (gates) из заблокированного состояния (locked state) в состояние готовности (ready state). Модуль управления блокировками (semaphore module) использует механизм защищенного сброса (secure reset), для которого перед сбросом требуется двукратная запись заданного шаблона в элемент блокировки (gate). Этот механизм требует, чтобы операции последовательной записи предопределенных данных выполнялись одним и тем же ядром (core). Такая последовательная запись проиллюстрирована в листинге 5, в котором показан пример реализации подпрограммы Reset_Gate. Подпрограмма сбрасывает один или все элементы блокировки (semaphore gates), в зависимости от значения номера элемента блокировки (gate number), переданного в подпрограмму Reset_Gate. Чтобы сбросить единственный элемент блокировки (gate), в подпрограмму необходимо передать номер этого элемента (gate). Чтобы сбросить все элементы блокировки, в подпрограмму Reset_Gate необходимо передать значение, большее, чем 63.

В листинге 5 приведен пример исходного кода на языке C для подпрограммы Reset_Gate. Эта подпрограмма пытается сбросить один или все элементы блокировки (gates). Для сброса отдельных элементов блокировки (gate) в модуль управления блокировками (semaphore module) последовательно передается два шаблона с установленными битами, соответствующими сбрасываемым элементам блокировки (gates).

В таблице 6 приведено описание кода подпрограммы Reset_Gate, представленного в листинге 5.

И в заключение рассмотрим примеры реализации классических семафоров[1] на основе

```
//-------------------------------------------------------------
// Reset_Gate
//-------------------------------------------------------------
1: uint8_t Reset_Gate (uint8_t gate_no)
2: {
3:      uint8_t cnt = 0;
4:      SEMA4.RSTGT.r = (0xE2<<8);
5:
6:      while (SEMA4.RSTGT.B.RSTGSM !=0x01)
7:      {
8:          if (cnt++>10) return 1
9:      }
10:
11:     SEMA4.RSTGT.R = (0x1D<<8) | gate_no;
12:
13:     return 0
14: }
```

Листинг 5. Подпрограмма Reset_Gate

Таблица 6. Описание кода подпрограммы Reset_Gate, представленного в листинге 5

№ строки	Описание
1	Подпрограмма Reset_Gate будет сбрасывать один или все элементы блокировки (gates) в соответствии с требованиями вызывающей процедуры: • для сброса одного элемента блокировки (gate): правильный номер элемента блокировки (gate number); • для сброса всех элементов блокировки (gates): номер элемента блокировки (gate number), больший, чем 63.
4	Запись первого шаблона данных (data pattern, 0xE2<<8) для получения доступа к механизму сброса элемента блокировки (gate).
6	Ожидание приглашения от конечного автомата (state machine) элемента блокировки (gate) записи следующего шаблона данных.
8	Если пройдено больше 10 полных циклов, подпрограмма будет находиться в состоянии ожидания (time out). (Конечный автомат (state machine) должен выдать ответ за время, не превышающее два периода частоты тактирования.)
11	Запись первого шаблона данных (data pattern, 0xID << 8) для получения доступа к механизму сброса и сброс элемента (всех элементов) блокировки (gate) в соответствии со значением, заданным параметром gate_no.

аппаратных элементов блокировки и функций управления ими.

Примеры классических семафоров

В разделе «Введение» упоминалось, что подпрограммы управления элементами блокировки (semaphore routines) являются неблокирующими. Это вызывает противоречия, поскольку классический семафор (semaphore) предназначен именно для выполнения функции блокирования. Однако неблокирующие подпрограммы управления элементами блокировки (semaphore routine), на которые обычно ссылаются как на подпрограммы семафоров (semaphore routine), позволяют приложению производить попытки захвата, или блокировки, аппаратного элемента блокировки (semaphore gate) без входа в бесконечный цикл (ожидание в цикле, пока постоянная проверка блокировки не выдаст результат, говорящий о доступности объекта). Если аппаратный элемент блокировки (semaphore gate) не может быть заблокирован, то функция вернет ошибку (fail) вместо блокировки; она не войдет в бесконечный цикл, пытаясь заблокировать элемент блокировки (gate). Если элемент блокировки (gate) может быть заблокирован, то модуль управления блокировками (semaphore module) заблокирует его и вернет в вызывающую процедуру значение true (элемент блокировки заблокирован).

Сигнальный семафор

Возможно, самым простым использованием семафора является сигнализация того, что одна задача посылает сигнал другой задаче, чтобы указать на какое-то происшествие. Сигнализация обеспечивает механизм, гарантирующий, что секция кода одной задачи выполняется перед секцией кода другой задачи.

В рассматриваемом примере задачи task 0 и task 1 будут выполняться на процессорных ядрах core 0 и core 1 соответственно. Обе задачи имеют совместный доступ (shared access) к некоему ресурсу. Задача task 0 записывает данные (writer), а задача task 1 считывает их (reader). Считывающая задача (reader) должна ожидать, пока записывающая задача (writer) не закончит запись информации в разделяемый (shared) ресурс и просигнализирует считывающей задаче (reader), перед тем как считывающая задача (reader) сможет прочитать информацию из разделяемого (shared) ресурса и ее отобразить.

В листингах 6 и 7 приведены примеры исходного кода на языке C для подпрограмм записи (writer) и чтения (reader), задачи task 0 и task 1 соответственно.

В примерах проиллюстрированы простейшие задачи записи (writer) и чтения (reader), использующие сигнальный семафор (signaling semaphore) для решения задачи сериализации (serialization problem) в многоядерной системе.

Таблица 7. Описание работы подпрограмм, представленных в листингах 6 и 7

№ строки	Подпрограмма записи Writer (задача task 0)	Подпрограмма чтения Reader (задача task 1)
3	Константа: CORE0_LOCK = 1	Константа: CORE0_LOCK = 1
6–9	Ожидание очистки (clear) флага семафора (semaphore flag) , разблокирование элемента блокировки gate 0 перед началом операции записи (write) в разделяемый ресурс (shared resource). Константа: Gate_0 = 0	Ожидание установки (set) флага семафора (semaphore flag), ожидание, пока задача task 0 просигнализирует о блокировании элемента блокировки gate 0 перед началом операции чтения (read) из разделяемого ресурса (shared resource). Константа: Gate_0 = 0
11	Запись (write) в разделяемый ресурс (shared resource).	–
13–17	Установка флага семафора (semaphore flag) — блокировка элемента блокировки Gate 0; сигнализирует задаче task 1, что информация готова для считывания.	–
11	–	(элемент блокировки Gate 0 заблокирован) Задача task 1 считывает (read) информацию из разделяемого ресурса (shared resource) и печатает строку текста.
13–17	—	Задача task 1 очищает (clear) флаг семафора (semaphore flag), используя подпрограмму Reset_Gate, и проверяет, разблокирован ли элемент блокировки gate 0.

```
//-------------------------------------------------------------
// Task 0 — Writer
//-------------------------------------------------------------
1: void Task0 (void)
2: {
3:      uint8_t status = CORE0_LOCK;
4:
5:      //wait for gate0 unlock
6:      while (status != UNLOCK)
7:      {
8:              status = Get_Gate_Status (GATE_0);
9:      }
10:
11:     sprint (buf, "%s\n", "Core 0 send this message");
12:
13:     while (status != CORE0_LOCK)
14:     {
15:
16:             status = Lock_Gate (GATE_0);
17:     }
18: }
```

Листинг 6. Сигнальный семафор: подпрограмма записи

```
//-------------------------------------------------------------
// Task 1 — Reader
//-------------------------------------------------------------
1: void Task1 (void)
2: {
3:      uint8_t status = CORE0_LOCK;
4:
5:      //wait for gate0 lock
6:      while (status != CORE0_LOCK)
7:      {
8:              status = Get_Gate_Status (GATE_0);
9:      }
10:
11:     sprint ("%s", buf);
12:
13:     while (status != UNLOCK)
14:     {
15:             Reset_Gate (GATE_0);
16:             status = Get_Gate_Status (GATE_0);
17:     }
18: }
```

Листинг 7. Сигнальный семафор: подпрограмма чтения

Следует отметить, что для разблокирования элемента блокировки gate 0 в этом примере вместо подпрограммы Unlock_Gate используется подпрограмма Reset_Gate перевода элемента блокировки GATE_0 в исходное состояние (release). Это сделано из-за того, что ядро core 0 заблокировало элемент блокировки gate 0, и нет никаких идей по поводу того, когда ядро core 1 будет считывать информацию, записанную ядром core 0. Кроме того, ядро core 1 не может разблокировать элемент блокировки (gate), заблокированный ядром core 0. Поэтому должен использоваться обходной путь Reset_Gate для перевода элемента блокировки gate 0 в исходное состояние (release), после того как ядро core 1 использовало информацию, записанную ядром core 0.

Блокирующий семафор

В этом примере используется симметричное решение для проблемы согласованности между двумя задачами, то есть обе задачи вызывают одну и ту же процедуру. Оба ядра, претендующие на блокировку элемента блокировки gate 0, будут вызывать функцию печати, чтобы получить доступ к устройству печати и напечатать строку текста. Первое ядро (core 0), чтобы заблокировать элемент блокировки gate 0, будет продолжать печатать строку текста и затем разблокирует элемент блокировки gate 0, перед тем как другое ядро (core 1) сможет заблокировать элемент блокировки gate 0.

Этот метод гарантирует, что функция принтера может напечатать из одной задачи сообщение целиком перед началом печати другого сообщения из другой задачи. Данный пример позволяет избежать потенциальной проблемы согласованности данных между обеими задачами.

В листинге 8 приведен пример исходного кода на языке C для подпрограммы печати (printer routine), иллюстрирующий простейший блокирующий (blocking semaphore), или двоичный семафор (binary semaphore), между двумя задачами и использующий симметричное решение. Принципы работы подпрограммы печати (printer routine) разобраны в таблице 8, где приведено описание кода подпрограммы печати, представленного в листинге 8.

Семафор в точке встречи

В сценарии рандеву (rendezvous) требуется, чтобы каждая задача, достигшая точки встречи (rendezvous point), сигнализировала о сво-

Таблица 8. Описание кода подпрограммы печати, представленного в листинге 8

№ строки	Ядро Core 0	Ядро Core 1
	Ядро Core 0 вызывает функцию печати --- Printer(CORE0_LOCK,msg0); Константа: CORE0_LOCK = 1 UNLOCK = 0	Ядро Core 1 вызывает функцию печати --- Printer(CORE1_LOCK,msg1); Константа: CORE1_LOCK = 2 UNLOCK = 0
6–9	Ядро Core 0 пытается заблокировать элемент блокировки gate 0 Константа: GATE_0 = 0	Ядро Core 1 пытается заблокировать элемент блокировки gate 0 Константа: GATE_0 = 0
	Предположим, что ядро core 0 заблокировало элемент блокировки gate 0	–
11	Ядро Core 0 печатает сообщение msg0	–
12–16	Ядро Core 0 разблокирует элемент блокировки gate 0	–
	–	Предположим, что теперь ядро core 1 заблокировало элемент блокировки gate 0
11	–	Ядро Core 1 печатает сообщение msg1
12–16	–	Ядро Core 1 разблокирует элемент блокировки gate 0

```
//-----------------------------------------
// Printer
//-----------------------------------------
1: void Printer (uint8_t core_lock, char *msg)
2: {
3:      uint8_t status = UNLOCK;
4:
5:      // try to lock gate 0
6:      while (status != core_lock)
7:      {
8:              status = Lock_Gate (GATE_0);
9:      }
10:
11:     Lpt1 (msg);
12:
13:     while (status == core_lock)
14:     {
15:             status = Unlock_Gate (GATE_0);
16:     }
17: }
```

Листинг 8. Блокирующий семафор: подпрограмма печати

ем присутствии и перед выполнением следующего шага кода ожидала, пока все задачи не достигнут точки встречи (rendezvous point). В рассматриваемом примере для решения задачи синхронизации используется семафор точки встречи (rendezvous semaphore).

В этом примере две задачи, выполняющиеся на ядре core 0 и ядре core 1, синхронизируются в точке встречи (rendezvous point). Когда одна из этих задач достигает точки встречи (rendezvous point), она должна выдать сигнал (блокировать элемент блокировки Gate n), чтобы оповестить другую задачу о своем прибытии. Когда две задачи прибывают в точку встречи (rendezvous point) и выдают сигналы, то обе задачи могут начинать выполнение следующего шага своего кода. В таком случае обе задачи будут синхронизированы.

В таблице 9 приведен пример исходного кода на языке C для семафора точки встречи (rendezvous semaphore). Этот пример иллюстрирует простейшее взаимодействие между двумя задачами в точке встречи (rendezvous) с целью их синхронизации. Исходный код примера детально разобран в таблице, чтобы подробно показать взаимоотношение между обеими задачами.

Классическая проблема: взаимная блокировка

Пример из предыдущего раздела «Семафор в точке встречи» («Rendezvous semaphore») можно переписать, изменив порядок строк исходного кода, чтобы создать взаимную блокировку (deadlock) между задачами task0 и task1 в точке встречи (rendezvous point).

В таком сценарии, когда задачи task 0 и task 1 достигают точки встречи (rendezvous point), нет ни одного из сигналов их присутствия, поскольку они находятся в состоянии ожидания того, что одна из задач просигнализирует первой. То есть задача task 0 ожидает, когда задача task 1 просигнализирует о достижении точки встречи, в то время как задача task 1 также ожидает, когда задача task 0 просигнализирует о достижении точки встречи. В таком случае налицо ситуация взаимной блокировки (deadlock). Это классическая проблема приложений, использующих концепцию семафоров. Имеются и другие классические и не классические проблемы, связанные с применением концепции семафоров. Подробности о возникновении таких проблем и методах их разрешения можно найти в многочисленной литературе, посвященной использованию семафоров.

Таблица 9. Описание поведения двух задач в точке встречи

Задача Task 0	Задача Task 1

//Задача Task 0
//достигла точки встречи:
// сигнализирует о присутствии.
// Константы:
// GATE_0 = 0
// GATE_1 = 1
// UNLOCK = 1
Lock_Gate (GATE_0);
	//Задача Task 1
	//достигла точки встречи:
	// сигнализирует о присутствии.
	// Константы:
	// GATE_0 = 0
	// GATE_1 = 1
	// UNLOCK = 1
	Lock_Gate (GATE_1);

//Check if all task 1 arrived?
status = UNLOCK;
while (status==UNLOCK)
{
Status = Get_Gate_Status (GATE_1);
}
	//Check if all task 0 arrived?
	status = UNLOCK;
	while (status==UNLOCK)
	{
	Status = Get_Gate_Status (GATE_0);
	}

Все задачи прибыли в точку встречи (rendezvous point) и сигнализируют о своем присутствии	
Задача Task 0 вызывает функцию A_Function ()	Задача Task 1 вызывает функцию B_Function ()

Заключение

В статье рассмотрены подпрограммы управления элементами блокировок Get_Core_ID, Get_Gate_Status, Lock_Gate, Unlock_Gate и Reset_Gate. Их исходный код на языке C предоставляет примеры того, как можно получить доступ к модулю управления блокировками (semaphore module) и сконфигурировать его. Эти подпрограммы также можно переносить в пользовательские приложения, без какой-либо доработки. Тем не менее, для тех, кто хочет написать собственные процедуры для управления аппаратными элементами блокировки (hardware semaphore), приведенные примеры могут существенно упростить задачу.

На этом заканчивается цикл статей, посвященных межпроцессорному взаимодействию в гетерогенных многопроцессорных СнК, одним из ядер которых является ядро ARM Cortex-M3/M4. Современные гетерогенные многопроцессорные СнК позволяют разделять ряд задач между ядрами различного типа с целью разгрузки основного ядра (основных ядер), на котором реализована бизнес-логика приложения, от вспомогательных задач, таких как управление вводом/выводом и выполнение специализированных вычислительных операций. Подобное разделение позволяет оптимизировать и упростить логику приложения, а также упорядочить потоки данных внутри кристалла СнК.

Автор надеется, что статьи будут полезны проектировщикам, выбирающим новые платформы для своих текущих задач.

Литература

1. AN4805. A Practical Approach to Hardware Semaphores For MCP56xx and MPC57xx Multicore Qorivva Devices. www.nxp.com/docs/en/application-note/AN4805.pdf
2. CodeWarrior embedded software development studio. www.nxp.com/support/developer-resources/software-development-tools/codewarrior-development-tools:CW_HOME