РЕАЛИЗАЦИЯ НОВЫХ МЕТОДОВ ВЫСОКОТОЧНЫХ ОЖИДАНИЙ ПРОЦЕССОВ ДЛЯ ОС LINUX

И.С. Федотова

1. Введние

Успешное выполнение высокопроизводительных вычислений, в частности на встроенных системах в промышленной сфере, всегда подразумевает наличие эффективного механизма замера времени и контроля временных прерываний. При разработке высокоточных технологий контроля и автоматизации промышленного оборудования возникает ряд требований и проблем. К ним относятся снижение использования ресурсов CPU, высокое разрешение замеров, эффективная обработка ожиданий и прерываний процессов.

В данной работе предлагается механизм обработки высокоточных временных интервалов как для ПК-платформы, так и для архитектуры встроенных систем ARM. Данное решение нацелено на разработку подхода унифицированного получения времени с аппаратного обеспечения ПК и таким образом заменяет известный способ измерения времени средствами системных вызовов ОС Linux. Хотя начиная с версии ядра Linux 2.6, произошли существенные изменения таймеров ядра, промах ожидания по прежнему составляет примерно 50 мкс. Поэтому часто для высокоточного сна используется метод ожидания в цикле на процессоре, что, с точки зрения параллелизма, не является эффективным. В рамках библиотеки таймерной поддержки *HighPerTimer* [1] [2], предлагается новый подход оптимизации сна, который снижает использование ресурсов CPU до 1-1,89%, относительно 100% ожидания в цикле на процессоре при сохранении минимальных потерь времени от точности пробуждения. Эта особенность расширяет сферу применения данной библиотеки и может быть реализована на языках выского уровня под ОС Linux процессорах ARM, x86 и x86-64.

2. Системный сон стандартной библиотеки языка С

Стандартная библиотека языка Си ОС Linux предоставляет возможность приостановить выполнение процесса на заданное количество секунд, микросекунд или наносекунд. Объявленный в заголовочном файле *unistd.h*, метод *sleep()* (подобно методам *usleep()* и *nanosleep()*) предоставляет простой способ заставить программу ждать заданный промежуток времени. Данные методы заставляют вызывающий его поток спать до тех пор, пока истечет заданное количество секунд (микросекунд или наносекунд) или до тех пор, пока поток получит определенный сигнал.

На рисунке 1 показана упрощенная схема исполнения системного ожидания. Очередь *Run-time*, (также известная как Готовый список *Ready list)*, хранит список всех процессов, которые готовы к выполнению и не заблокированы на операциях I/O или других системных запросах, как например семафор. Записи в списке являются указателями на блок управления процессами, который хранит всю информацию о каждом состоянии. Когда выполнение какого-то процесса приостанавливается, сначала он попадет в очередь *Run-time*. После этого, в соответствии с его приоритетом, процесс попадает в Очередь ожидания (*Waiting queue*), где ждет установленное время. После того как время сна истекло, процесс переходит из состояния ожидания в состояние готовности, попадает в список *Ready list* и завершает свое исполнение [3]. Вся эта рутина контролируется планировщиком процессов.

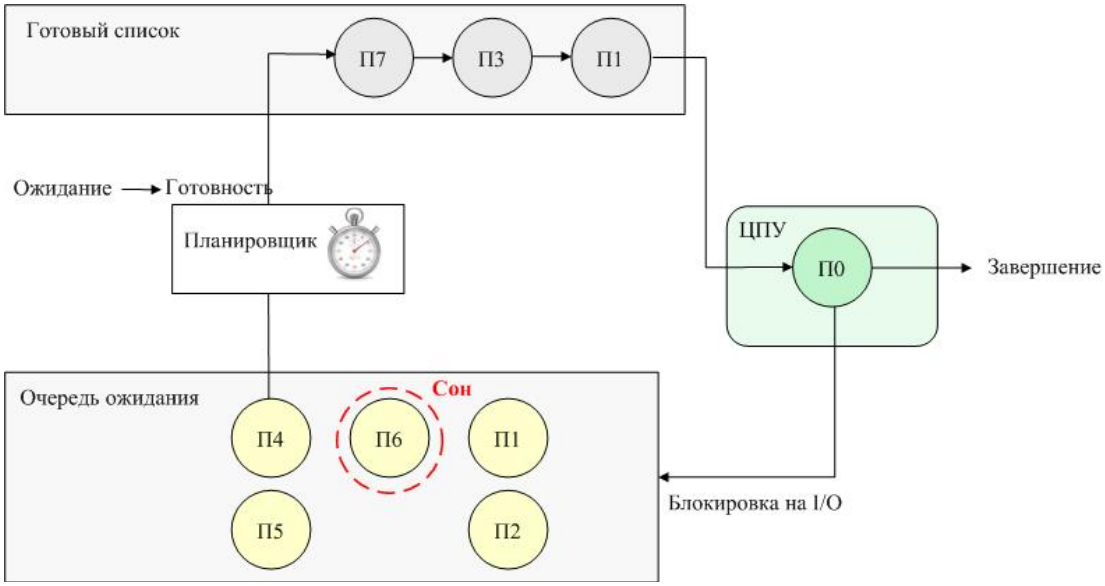

Рис. 1. Упрощенная схема выполнения процессов в Linux

Планировщик процессов является компонентом операционной системы и решает, какие задачи или процессы будут добавлены в очередь процессов, готовых к выполнению. Планировщик либо добавляет новый процесс в очередь готовых процессов (допускает к выполнению), либо откладывает это действие. Планировщик также «будит» спящие процессы [4]. Он функционирует с частотой HZ, а это означает, что планировщик может проверить состояние процесса в очереди не чаще, чем 1/HZ секунд. HZ (hertz) – это частота ядра, на которую настроен таймер системы для прерываний, определяется как количество тактов в секунду. Прерывания таймера представляют собой механизм, используемый ядром для получения требуемых интервалов времени. Значение HZ устанавливается во время компиляции ядра. Современный Linux использует прерывания с частотой 1000 Гц, но значения варьируются в зависимости от версии ядра и аппаратных платформ и могут быть также равны 300, 250 или 100.

Таким образом, выполнение ожидания процесса происходит без использования ресурсов процессора, что является основным преимуществом системного сна. Однако, промах сильно зависит от значения HZ и версии ядра, что является основным недостатком данного метода. Под промахом имеется в виду, отклонение или разница реального времени сна от ожидаемого. На рисунке 2 показаны результаты промаха системного сна при задержке от 1 с до 1 мкс. Измерения проводятся на процессоре Intel Core-i7 с HZ = 1000.

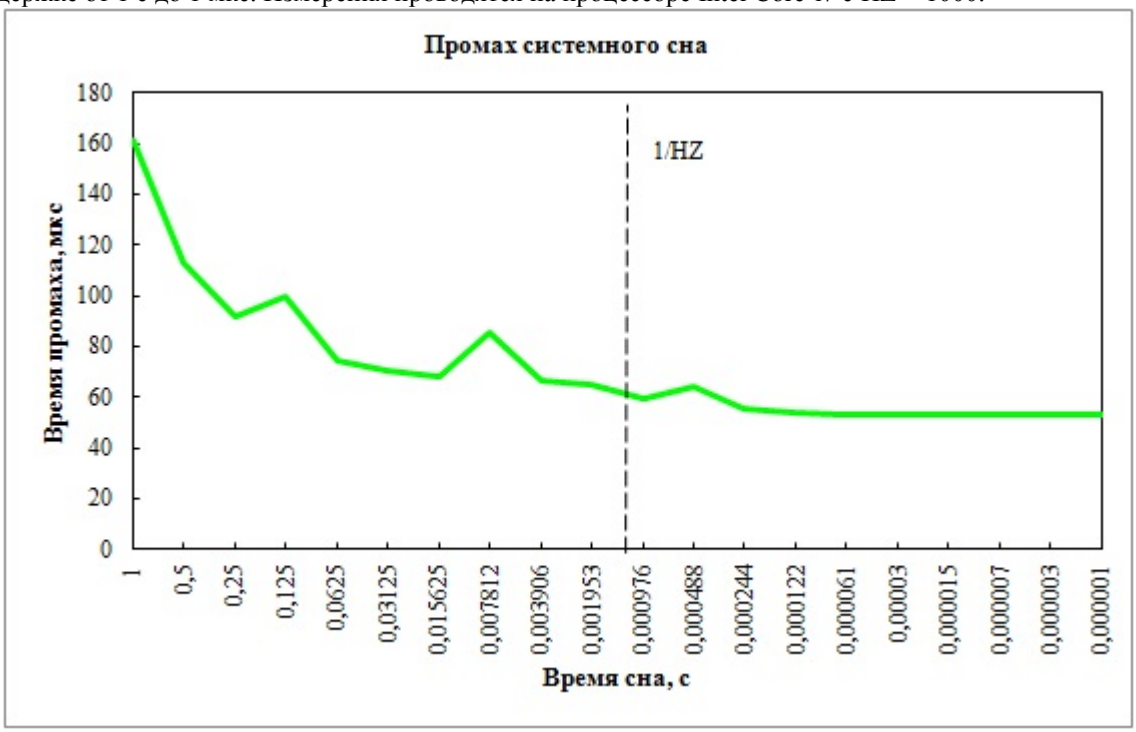

Рис. 2. Промах системного сна

3. Сон в цикле ожидания на процессоре

Альтернативный способ исполнения сна - ожидание в цикле на процессоре (*busy-waiting loop*). При таком подходе процесс ожидает события, вращаясь всё время в «плотном» цикле. Это позволяет сэкономить время и уменьшить промах, но недостаток данного подхода в том, что во время сна мы имеем 100% загрузку CPU. В спецификации Intel рекомендуется использовать ассемблерную инструкцию *pause* [5]. На более старых процессорах, эта инструкция работает как *nop*. Инструкция *nop* не выполняет никаких действий, и обычно используется вместе с инструкцией *rep*. Кроме того, например, на ARM архитектуре не поддерживается ни *rep* ни *pause* инструкции, поэтому в этом случае возможно использование только *nop*. Такой способ использует 100% производительности CPU, но в соответствии с рисунком 3 имеет очень маленькое значение промаха. Измерения проводятся на процессоре Intel Core-i7 с HZ = 1000.

Рис. 3. Промах ожидания в цикле на процессоре

Следует также заметить, что значение промаха выравнивается после времени сна равному 0,001953 с и становится равным примерно 50 нс. Значения промаха здесь также зависят от величины 1/HZ. Когда время сна превышает параметр 1/HZ, значение промаха уменьшается прямо пропорционально к запланированному времени сна. Когда время сна меньше, чем 1/HZ, величина промаха равна примерно 50 нс, что вполне приемлемо для высокоточного сна.

4. Предлагаемый метод оптимизации ожиданий

HighPerTimer сон является новым комбинированный способом сна, который объединяет выполнение системного сна и ожидания в цикле на процессоре. При его разработке ставилась задача получить преимущества от использования каждого способа, т.е. минимальную загрузку процессора и минимальный промах. Основная идея заключается в разделении общего времени сна на две части, используя величину 1/HZ.

Так как Linux не предлагает стандартное решение запроса значения HZ, используемое ядром, это значение приходится определять опосредованно. В качестве метода расчета HZ был выбран способ оценки загрузки процессора во время ожидания в цикле. Анализируя, как ядро замеряет время использования ресурсов процессора на разных платформах, были обнаружены некоторые зависимости. Каждый таймер имеет свое собственное разрешение (точнее, это разрешение равно 1/HZ). Например, при HZ = 1000 разрешение будет равно 0,001 с, а измерить использование ресурсов CPU проще, когда значение времени сна составляет 0,01 с, 0,02 с, 0,03 с и так далее, т.е. кратно разрешению. Следовательно, на такой платформе во время ожидания в плотном цикле на 0,001 с, время использования ресурсов процессора и будет равно 0,001. А если заставить процесс спать на время не кратное 1/HZ, а именно на 0,0025 с, то время использования ресурсов CPU будет практически всегда равняться нулю, и изредка 0,01 с. Это означает, что значения времени сна, больше или меньше значений кратных 1/HZ, не являются “значемыми” и по ним сложно сделать корректные измерения. Чтобы упростить и ускорить расчет величины HZ на всех видах платформ за раз, было выбрано компромиссное значение времени сна - 14500 мкс. С помощью этого значения можно различить и идентифицировать HZ конкретной платформы быстрее. На рисунке 4 объединены результаты замера использования системных ресурсов на различных платформах, но при едином значении времени сна. Заметно, что значения сильно варьируются, однако каждая линия (соответствующая отдельно взятой платформе) имеет одно значение, которое встречается чаще. Для величины HZ равной 100 (красная линия), доминирует значение времени использования ресурсов CPU 0,01 с (или менее вероятное 0,02 с), для HZ = 250 (зеленая линия) - это 0,016 с (или реже 0,012 с), для HZ = 300 (синяя линия) - это 0,0133 с (или реже 0,016 мс), для HZ = 1000 (малиновая линия) — 0,014 с (или менее вероятное 0,015 с).

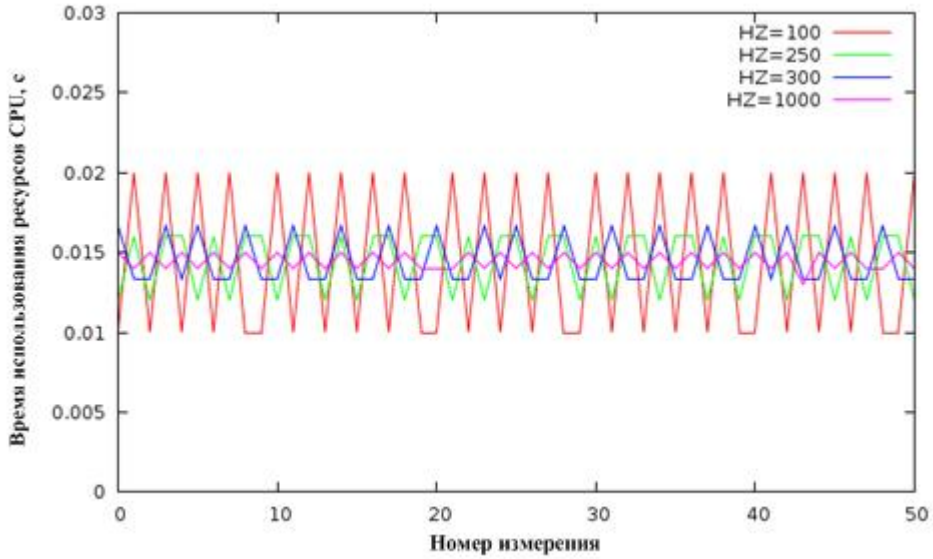

Рис. 4. Измерения загрузки CPU во время ожидания на процессоре на 14500 мс на разных платформах

Наиболее часто встречающиеся значения объеденненны в таблице 1:

Таблица 1. Значения использования ресурсов CPU во время ожидания на процессоре на 14500 мс

HZ, прерываний/с	Использование CPU, мс
100	10
250	16
300	13.3
1000	14

Таким образом, на стадии инициализации библиотеки происходит анализ конкретной платформы и величины HZ для дальнейшей реализации методов ожидания.

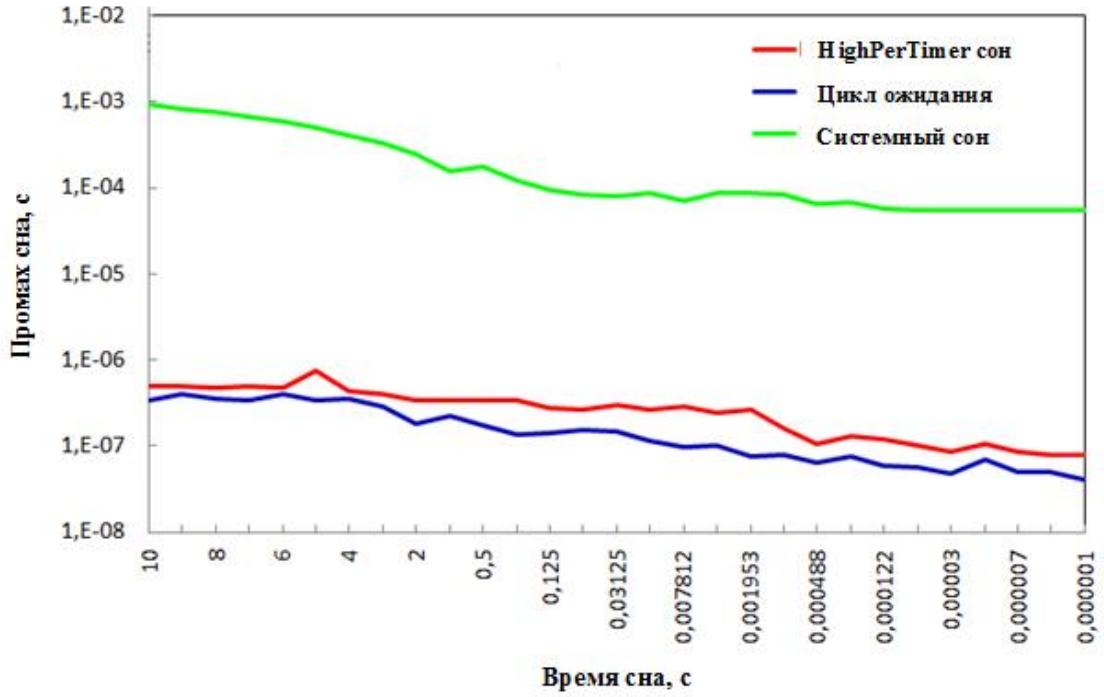

Рис. 5. Измерение зависимости значения промаха от времени сна

На рисунке 5 показаны результаты измерения значения промаха в зависимости от времени сна, которое варьируется от 10 с до 1 мкс. Когда время сна больше чем 1/HZ, наблюдается тенденция зависимости промаха от времени сна. Более того, подобная тенденция наблюдается как во время системного сна, так и и во время ожидания в цикле на процессоре. В отличие от этого, в интервале, когда время сна меньше, чем 1/HZ, промах остаётся нетеденциально константным и колеблется в диапазоне от 50 до 100 нс.Время сна>=1HZ.

В таблице 2 показаны результаты измерений среднего значения промаха при каждом методе ожидания. Для измерения был использован цикл из 10000 шагов, на каждом шаге которого были произведены замеры на время сна от 1 с до 1 мкс. Общее время теста составило 830 минуты.

Таблица 2. Оценка величины промаха всех известных методов сна

		Системный сон	Цикл ожидания	HighPerTimer сон
Время сна>=1HZ	Мат ожидание, мкс	61,9	0,160	0,258
Время сна>=1HZ	Мат ожидание, мкс	50,9	0,0701	0,0950
	Стан. Отклонение, мкс	11,4	0,0400	0,0404

В интервале когда временя сна больше, чем величина 1/HZ, в расчете стандартного отклонения нет смысла, т.к. промах зависит от времени сна тенденцией проиллюстрированной на рисунке 5. Когда время сна меньше, чем 1/HZ, значение промаха при выполнении сна *HighPerTimer* составляет в среднем 95 нс. Значение стандартного отклонения на этом интервале составляет около 404 нс. Кроме того, необходимо оценить *HighPerTimer* сон по параметру загрузки CPU. В таблице 3 представлены результаты процентного отношения использования процессорного времени в сравнении с системным сном и циклом ожидания:

Таблица 3. Оценка загрузки процессора во время выполнения ожидания

	Системный сон	**Цикл ожидания**	***HighPerTimer* сон**
Реальное время выполнения	835 мин 49,698 с	833 мин 18,146 с	833 мин 18,675 с
Время использования CPU	8,179 с	830 мин 38,679 с	15 мин 46,006 с
Процентное отношение	0,0160%	99,7%	1,89 %

Измерения проводились в цикле из 10000 шагов, время сна на каждом шаге варьируется от 0,25 с до 1 мкс, при этом реальное время теста составило более 833 минут для каждого метода. Из таблицы заметно, что во время исполнения сна в цикле ожидания на процессоре, загрузка CPU равна 99,7%. В сравнение с этим значением, загрузка процессора в 1,89% при выполнение *HighPerTimer* сна выглядит существенным преимуществом.

Анализируя результаты всех сделанных измерений, можно сделать вывод, что *HighPerTimer* сон обладает высокой точностью и подходит для высокопроизводительных вычислений. На интервале, когда время сна меньше, чем 1/HZ, значение промаха равно примерно 95 нс. Кроме того, основное преимущество использования *HighPerTimer* сна заключается в практически полной независимости от количества процессорных ядер. Процент использования ресурсов CPU минимальный и составляет только 1,89%.

5. Обработка прерываний нового метода сна

С тех пор как библиотека *HighPerTimer* обладает механизмом обработки ожидания процессов, появилась необходимость создания соответствующего механизма для прерывания этого спящего состояния. Прервать спящий процесс означает остановить его действие в определенный момент времени с максимально возможной точностью. Механизм прерывания процесса реализован в рамках метода библиотеки *HighPerTimer::Interrupt()*.

Условно задача прерывания делится на две части — прерывания процесса в момент нахождения в цикле ожидания и в момент выполнения системного сна. Реализация первой части достаточна тривиальна. При вызове метода прерывания, соотвествующий флаг прерывания (член экземпляра класса логического типа) меняет свое значение на «истина». Далее в цикле ожидания, проверяется значение флага, происходит выход из цикла и сон завершается. Однако, нет смысла проверять значение флага на каждом шаге, более того следует избегать слишком частый доступ к переменной. Для этих целей было выбранно компромиссное значение, т. е. флаг прерывания проверяется только на каждом 16ом шаге.

Задача прерывания процесса в момент нахождения в системном сне, решается с помощью методов библиотеки <*thread*> нового стандарта языка C++11. В новой версии стандарта добавленна поддержка многопоточности, что позволило реализовать концепцию потоковой безопасности (*thread-safety*) и избежать ошибок в условиях возможной потоковой гонки (*race condition*) [6]. Использование мьютексов помогает определить критические области в коде, точнее регионы проверки значения флага прерывания потока, а более эффективная синхронизация обеспечивается благодаря использованию условных переменных. Пример использования методов библиотеки *HighPerTimer* приведен в Листинге 1:

```
// create HighPerTimer object
HPTimer::HighPerTimer MyTimer;

// interrupt thread
std::thread MyThread([this]()
{
      // invoke interrupt method for sleeping thread
      this->Timer1.Interrupt();
}
);

// sleeping thread
// sleep for 50 microseconds
MyTimer.USecSleep(50);

MyThread.join();
```

Листингг 1. Вызов метода прерывания из другого потока

6. Заключение

Разработанное решение имеет высокий потенциал применения в сфере телекоммуникационных технологий и в области промышленности в целом. Оно позволяет улучшить оценку высокопроизводительной работы программных приложений, коммуникационных процессов, сделать анализ многих производных параметров сети более точным.

Одним из последующих шагов является улучшенная поддержка системного таймера ARM процессоров (одноядерных на базе семейств Cortex-A8 и двухъядерных на базе Cortex-A9), на которых отсутствуют известные для ПК-платформ счетчики TSC и HPET [2]. Согласно документации, реализация ARM должна поддерживать системный таймер GP Timer (Global Purpose Timer), отображенный в «закрытую» область памяти и значения которого могут быть считаны только из режима ядра [7][8]. Предположительно, прямое обращение к данному системному таймеру может позволить сэкономить несколько дополнительных микросекунд и существенно улучшить производительность выполнения ожиданий процессов.

ЛИТЕРАТУРА:

1. I. Fedotova, E. Siemens, H. Hu. “A high-precision Time Handling Library” // Proc. of: The Ninth International Conference on Networking and Services, (ICNS 2013), Lisbon, March 2013, pp. 193-199.
2. I. Fedotova, E. Siemens. “Self-configurable time source initialization for obtaining high-precision user-space timing” // Vestnik SibSUTIS, 4.2012, Novosibirsk, Russia, Nov. 2012, pp. 22-30.
3. M. Kravetz, H. Franke, S. Nagar, R. Ravindran, “Enhancing Linux Scheduler Scalability” // Proc. of: The Ottawa Linux Symposium, Ottawa, Canada, Jul. 2001.
4. D. Bovet, M. Cesati. “Understanding Linux Kernel” // O'Reilly. Second edition, 2003, Ch. 10.2.
5. Intel 64 and IA-32 Architectures, Software Developer‘s Manual, vol. 1 11-18. URL: http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-software-developer-vol-1-2a-2b-3a-3b-manual.pdf (дата обращения 23.04.2013).
6. S. Meyers “Overview of the New C++ (C++11)” // Artima Press. Fifth edition, 2012 // Slide 91.
7. Cortex-A8, revision: r3p2, Technical Reference Manual. Chapter 3, pp. 3-83. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344h/Bgbjjhaj.html.
8. Cortex-A9, revision: r4p0, Technical Reference Manual. Chapter 4, pp 4-1. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0407f/CIHGECHJ.html.