УДК 004

Исследование эффективности аппаратной реализации отслеживания зависимостей по данным в структуре конвейера сопроцессора СР2 микропроцессора КОМДИВ128-RIO

А.В. Тёма, С.В. Белобородова

Учреждение Российской академии наук Научно-исследовательский институт системных исследований РАН, tyoma@niisi.msk.ru

***Аннотация* — Статья посвящена исследованию эффективности аппаратной реализации отслеживания зависимостей по данным в структуре конвейера сопроцессора вещественной арифметики процессора КОМДИВ128-RIO. Рассматриваются виды конфликтов по данным, способы их устранения. Определяется влияние динамического отслеживания зависимостей на производительность процессора, отладку программ, разработку программного обеспечения.**

***Ключевые слова* — конфликты по данным.**

I. Введение

Микропроцессор КОМДИВ128-RIO, разрабатываемый в НИИСИ РАН в настоящее время, предназначен для обработки сигналов и содержит в своем составе специализированный сопроцессор СР2 для ускорения вычислений над комплексными числами и числами с плавающей точкой. Сопроцессор СР2 работает на частоте 250МГц, имеет SIMD архитектуру, четыре вычислительные секции; пиковая производительность на операциях с 32-разрядными вещественными числами достигает 8 ГФлопс. В настоящей реализации специализированный сопроцессор СР2 не содержит аппаратных средств для аппаратного отслеживания конфликтов по данным: проблемы по выявлению и устранению данных конфликтов полностью возложены на разработчиков программного обеспечения.

Исследование проводилось с целью определения эффективности аппаратного отслеживания конфликтов по данным в структуре конвейера сопроцессора СР2, так как данные конфликты являют собой большую проблему для разработчиков и аппаратуры, и программного обеспечения. Исследование проводилось на классе задач стандартной обработки сигналов с использованием потактовой поведенческой СИ-модели и RTL-модели микропроцессора КОМДИВ128-RIO.

Проблема зависимости по данным возникает в процессорах и микропроцессорах при использовании конвейера для исполнения инструкций и состоит в том, что если последующая команда использует результат предыдущей, то последняя не может начать выполняться до окончания выполнения первой. Конфликты по данным бывают трех типов:

1) RAW – Read After Write**.** Данный конфликт возникает в том случае, когда команда процессора осуществляет считывание из регистра, который считается недоступным. Недоступным считается тот регистр, в который еще не осуществлена запись любой предшествующей командой. Данный тип конфликтов может возникнуть на стадии чтения операндов и может быть разрешен только с помощью ожидания результата предшествующей команды.
2) WAR – Write After Read**.** Данный конфликт возникает в ситуации, когда операция записи в регистр предшествует чтению значения из этого регистра командой, идущей раньше по цепочке конвейера. Поэтому операция записи в регистр откладывается до тех пор, пока все предыдущие инструкции не пройдут этап чтения. Данный конфликт может возникнуть на стадии записи результатов. Устраняются такие конфликты путем резервного сохранения «старого» значения регистра.
3) WAW - Write After Write**.** Такие конфликты возникают в том случае, когда две или более инструкций выполняют запись в один и тот же регистр. Конфликты такого типа фиксируются на стадии записи результатов и могут быть устранены путем ввода различных тегов для регистров назначения.

II. Проблематика. Факторы эффективности

Проблема зависимости по данным возникла с появлением конвейера в процессоре. С тех пор были предложены различные алгоритмы (Scoreboard, Tomasulo),

написаны особые компиляторы для решения данного вопроса. К настоящему моменту существуют стандартные алгоритмы и аппаратура для отслеживания и устранения зависимостей по данным. Но так было не всегда. История знает ошибочные структуры и решения.

С учетом стоимости и размеров транзисторов 80-х годов каждая фирма стремилась к сокращению объема аппаратуры, многие задачи предполагалось решать с помощью прикладного программного обеспечения [1] - [2]. Стало ясно, что зависимость по данным можно устранить двумя путями – аппаратно или программно, но от первого способа отказывались по вышеупомянутым причинам. Это объясняет плачевную историю первого процессора MIPS. Процессор MIPS-X имел стандартный пятиступенчатый конвейер. Предполагалось, что специальный компилятор будет вставлять NOP между двумя инструкциями, зависящими по данным между собой, и таким образом простоев конвейера не будет. При выполнении многоэтапной инструкции типа деления, вставленные NOP'ы увеличивали бы время исполнения команды во много раз. Поэтому разработчики первого микропроцессора MIPS-X исключили из набора команд деление и команды для обработки чисел с плавающей точкой. Такой микропроцессор просуществовал недолго. Разработчики быстро устранили недостаток и выпустили в 1984 году R2000, а позднее и R3000[2].

В настоящее время проблема аппаратного отслеживания зависимостей по данным решена уже во многих специализированных процессорах, как отечественных, так и зарубежных аналогах.

Таблица 1

Аппаратная реализация отслеживания конфликтов по данным в современных процессорах

Процессор, фирма-разработчик	**Наличие аппаратной реализации отслеживания зависимости по данным**
Отечественные специализированные процессоры	
«Мультикор» МС-24 , «Элвис» [3]	+
«Мультикор» МС-12, «Элвис» [3]	+
«Мультикор» МС-0226, «Элвис» [3]	+
«NeuroMatrix», НТЦ «Модуль» [4]	+
Зарубежные специализированные процессоры	
C-6000, «Texas instruments»[5]	+
TigerShark-201, «Analog Devices»[6]	+
Процессоры серии ARM11[7]	+

Для определения эффективности динамического отслеживания конфликтов в процессе исследования были выделены основные факторы, на которые в значительной степени влияет наличие/отсутствие аппаратных средств отслеживания зависимостей по данным. К данным факторам относятся:

1) Объем введенной аппаратуры в процессор.
2) Производительность процессора.
3) Уровень сложности отладки программ на реальном процессоре.
4) Уровень сложности разработки программ.
5) Степень совместимости программ при модернизации конвейера.

В процессе исследования была проведена оценка каждого из факторов.

III. Объем введенной аппаратуры

В настоящей реализации сопроцессор СР2 микропроцессора КОМДИВ128-RIO не имеет специальных аппаратных средств для отслеживания зависимостей по данным. Контроль возлагается только на разработчиков программного обеспечения.

Для динамического отслеживания конфликтов в сопроцессор СР2 процессора К128-РИО предполагается внесение дополнительной аппаратуры в виде блока Scoreboard.

На RTL-модели процессора было проведено исследование, как дополнительный блок повлияет на характеристики процессора.

В режиме аппаратного отслеживания зависимостей сопроцессор отслеживает порядок исполнения взаимозависимых инструкций. Например, если инструкции в стадии декодирования, требуется результат предыдущей команды, находящейся в стадии исполнения, то выполнение текущей команды будет приостановлено до завершения предыдущей команды.

Для уменьшения времени простоя конвейера применяется механизм «bypass», заключающийся в том, что результат вычислений команды, находящейся на последней стадии конвейера, может быть использован текущей командой, до того как он будет записан в регистровый файл.

Длительность задержки latency (с учетом механизма bypass) исполнения зависимой команды зависит от длины конвейера, и в данном случае составляет 7 тактов для арифметических команд, 1-2 такта для операций пересылок и 1 такт для команд управления.

Каждая из 4 вычислительных секций сопроцессора СР2 содержит 64 регистра FPR для обработки чисел с плавающей точкой. В соответствии с алгоритмом формируется интерактивная таблица, которая содержит значения счетчиков простоя для каждого из 64 регистров. Счетчики простоя содержат количество тактов, в течение которых данный регистр будет недоступен для чтения. В конвейере для команды, желающей считать результат из недоступного регистра, выставляются вынужденные простои.

N команды	Номер такта															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CMUL f2, f1, f0	I	U	M1	M2	A1	A2	A3	A4	WB							
Latency_counter_f2	0	7	6	5	4	3	2	1	0							
Busy_f2	0	1	1	1	1	1	1	1	0							
CMUL f3, f1, f2		I	stall	stall	stall	stall	stall	stall	U	M1	M2	A1	A2	A3	A4	WB
Latency_counter_f3		0	7	6	5	4	3	2	1	0						
Busy_f3		0	1	1	1	1	1	1	1	0						
CMUL f4, f5, f6			stall	stall	stall	stall	stall	stall	I	U	M1	M2	A1	A2	A3	A4
Latency_counter_f4									0	7	6	5	4	3	2	1
Busy_f4									0	1	1	1	1	1	1	1
CMUL f9, f3, f2				stall	stall	stall	stall	stall	stall	I	stall	stall	stall	stall	stall	U
Latency_counter_f9										0	7	6	5	4	3	2
Busy_f9										0	1	1	1	1	1	1

Рис. 1. Конвейер CP2. Пример возникновения зависимости по данным

Пример показан на рис. 1. После декодирования первой команды, счетчик для регистра-приемника f2 принимает значение latency команды CMUL, сигнал Busy становится = 1 и держится до обнуления счетчика. Все это время регистр F2 для чтения недоступен.

На втором такте конвейер выбирает команду 2, которая должна считать значение из регистра f2, но на считывание регистра f2 стоит сигнал Busy, поэтому в конвейер добавляются вынужденные простои.

Маршрут обработки инструкций, поступающих на вход CP2 представлен на рис. 2. Схема обнаружения зависимостей по данным включена в управляющий блок Cunit. На основе сигнала Busy, выдаваемого этой схемой, блок Cunit выставляет управляющий сигнал, поступающий на входы всех секций.

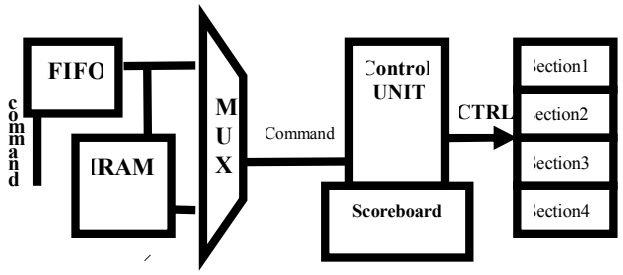

Рис. 2. Блок-схема продвижения команды в CP2

Рассмотрим подробнее схему scoreboard. Для каждого регистра FPR заводится свой счетчик (C0-C63). Регистровый файл сопроцессора CP2 имеет 3 порта на запись и 4 порта на чтение, которые необходимы для того, чтобы и команда пересылки, и арифметическая команда могли считывать свои регистры-источники без задержки. Каждый счетчик выдает сигнал Rbusy, значение которого зависит от входящих сигналов. Каждый сигнал Rbusy поступает на вход четырех мультиплексоров (по количеству портов на чтение регистров-источников). Также на входы мультиплексоров подаются сигналы Raddr[5:0], содержащие номер регистра-приемника, необходимого для следующей команды. Если какой-либо из сигналов Rbusy счетчика C0-C63 установился в 1, а также Raddr=1 для этого регистра, то на выходе мультиплексора устанавливается 1. Далее идет проверка разрешения чтения по данному порту (сигналы en_read0,1,2,3 для каждого порта), чтобы удостовериться в том, что чтение действительно произойдет. Если оба сигнала установлены в 1, то на выходе блока «ИЛИ» получаем сигнал busy 0, 1, 2 или 3, равный 1. Последний блок «И» анализирует все четыре сигнала busy0, busy1, busy2 и busy3 и выставляет общий сигнал Busy.

А. Критический путь №1

Критическим путем будем называть путь, проблематичный с точки зрения временных задержек. Частота работы сопроцессора равна 250 МГц (длительность такта = 4 нс). В связи с увеличением количества логики, осуществляющей механизм bypass и отвечающей за выдачу сигнала Busy, на стадиях конвейера **I** (выборка и декодирование команды) и **U** (распаковка и коммутация данных из регистрового файла) возникают дополнительные задержки.

Первый путь связан с большим временем адресации регистрового файла. Для каждого из 4-х портов чтения операнда инструкции вводится сигнал Raddr[5:0].

Сигнал Raddr[5:0], указывающий адрес регистра, необходимого для текущей инструкции, должен поступить на вход регистрового файла на 1-ой стадии конвейера. Этот сигнал выдает блок CUnit, который декодирует поступившую на его вход команду и в соответствии с ее форматом вычисляет значения адресов регистров (рис. 3).

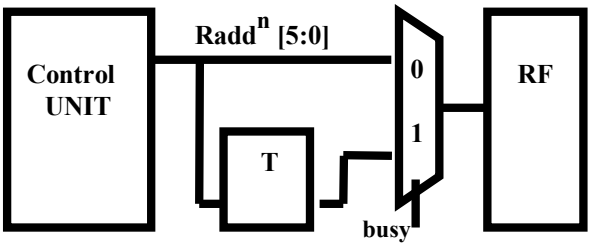

Рис. 3. Адресация регистрового файла

Данный сигнал на вход регистрового файла (RF) через мультиплексор. В случае, когда на мультиплексор подан сигнал Busy = 1, адрес защелкивается через триггер. Одной из проблем в данном случае является медленный доступ к регистровому файлу. Сигнал Busy также выставляется поздно за счет задержек от мультиплексоров схемы scoreboard. В общем, получаем задержку в 5 нс, которая недопустима при заданной частоте работы сопроцессора. С учетом расположения элементов на кристалле, при физическом синтезе возникают дополнительные задержки в 1-2 нс.

Б. Критический путь №2

Второй критический путь связан с чтением данных из регистрового файла на стадии «U» конвейера и передачей данных по bypass. Главной проблемой является умножитель M1 блока FPU, данные на который нужно подать еще на стадии «U». Но данные, считанные из регистрового файла, проходят через несколько мультиплексоров, прежде чем записываются в FPU, а это приводит к большим задержкам порядка 400 пс, и данные не поступают вовремя.

При реализации механизма bypass, данные считываются напрямую, когда запись в регистровый файл еще не прошла. При этом выигрывается один такт. Данный механизм реализуется с помощью 4 мультиплексоров (по количеству портов на чтение) для правильного коммутирования данных, поступающих в FPU. На вход мультиплексоров подаются 64-разрядные данные и номер порта для записи. Каждый мультиплексор имеет 3 шины управления (рис. 4). Из-за большой разрядности данных, подаваемых на вход мультиплексора, появляется задержка в 1нс. Стадия «U» даже без механизма bypass является сложной, так как в FPU стоит большой коммутатор данных, поступающих в FPU: из ПЗУ, из глобальных регистров.

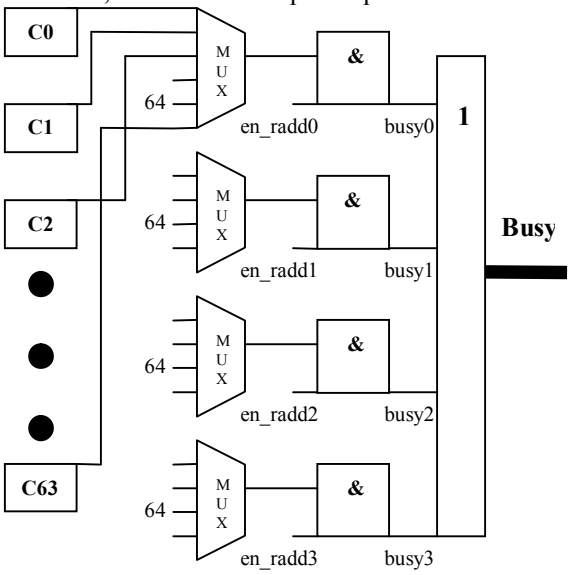

Рис. 4. Схема Scoreboard в CP2

Таблица 2

Выводы по внесению дополнительного оборудования в процессор

Фактор	Изменение фактора*	Причина
Длительности стадий «I» и «U» конвейера	Увеличиваются. Реально имеет ситуация, что данные стадии длятся дольше, чем один такт.	Внесение дополнительной аппаратуры, в частности триггеров и мультиплексоров, на которых увеличиваются задержки
Длина трасс	Возрастает. Трассы усложняются. Возникают дополнительные емкости и, соответственно, дополнительные задержки.	Внесение дополнительной аппаратуры, в частности триггеров и мультиплексоров
Площадь кристалла	Увеличивается на 5-10%	Внесение дополнительной аппаратуры
Энергопотребление	Увеличивается на 5-10%	Внесение дополнительной аппаратуры, удлинение трасс
Производительность	Падает на 10%	Внесение дополнительной аппаратуру, в частности дополнительных триггеров и мультиплексоров, а которых возникают дополнительные задержки

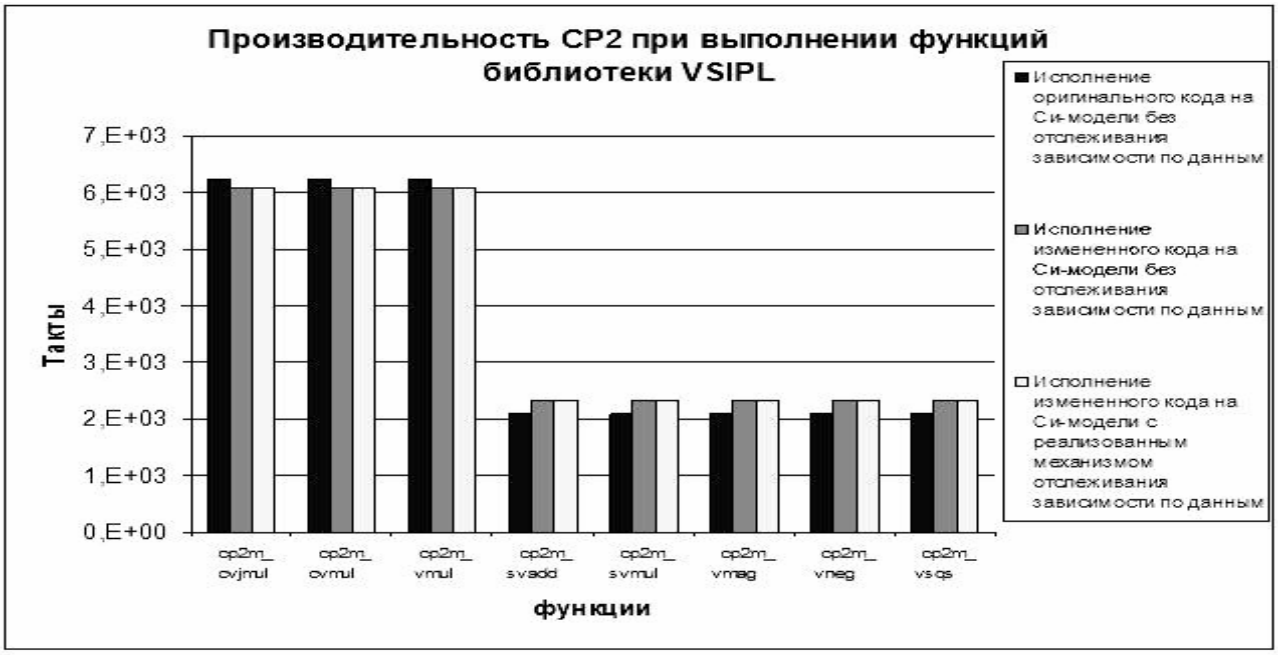

Рис. 5. Производительность CP2 при выполнении функций стандартной библиотеки VSIPL

IV. Программное обеспечение. Анализ производительности программ

Разработка программного обеспечения для процессора, не отслеживающего аппаратно зависимости по данным в несколько раз сложнее, чем для процессора с динамическим отслеживанием конфликтов, потому как, например, в случае с сопроцессором CP2 программисту придется учитывать длину двух конвейеров (арифметический конвейер и конвейер операций управления и пересылок) разной длины и самому выверять зависимости. В условиях, когда каждая лишняя команда стоит значительной потери производительности программы, разработка программного обеспечения гораздо усложняется.

Для сопроцессора CP2 в НИИСИ РАН была разработана библиотека VSIPL - Vector, Signal and Image Processing Library - содержащая набор стандартизированных функций, предназначенных для использования в программах обработки сигналов. Данная библиотека рассчитана на процессор, не имеющий аппаратных средств отслеживания зависимостей по данным, и использует особенности архитектуры для увеличения производительности программ сопроцессора.

В процессе исследования выяснялась возможность создания универсального программного обеспечения, корректно исполняющегося и на процессоре с динамическим отслеживанием конфликтов, и без такового; также определялась производительность программ библиотеки VSIPL на двух вариантах процессоров.

Исследование проводилось на СИ-модели микропроцессора КОМДИВ128-RIO, в которой для сопроцессора CP2 предусмотрено 2 режима работы: с динамическим отслеживанием конфликтов и без такового.

Главный интерес представляют собой функции, разработанные на основе 8-ми или 16-ти потоков данных. Кратность 8 выбрана потому как арифметическая команда требует 8 тактов до помещения результата в регистровый файл.

Исследование показало, что возможна модификация библиотеки для корректного исполнения в двух режимах процессора. Результаты модификации показаны в таблице. Функции, основанные на обработке 8 потоков данных, используют в первоначальном варианте особенности архитектуры и поэтому при модификации требуют на 12,5% больше времени на выполнение. Функции, основанные на обработке 16 потоков данных, после модификации удается оптимизировать и добиться выигрыша в производительности порядка 5,5%. Важно отметить, что 8-ми поточные функции можно переложить на 16 потоков и также получить выигрыш в производительности (рис. 5).

V. Отладка программ на реальном процессоре

Отладка программы подразумевает остановку её выполнения, последующую проверку содержимого регистров процессора, выполнение дополнительных проверочных команд. Все эти действия вызывают изменение состояния конвейера после остановки.

При реализации в процессоре аппаратного отслеживания зависимости по данным все выбранные ранее команды проходят по конвейеру до стадии WB (запись результат вычислений в регистровый файл), и результаты выполнения пройденных команд записываются в заданные регистры. После приостановки программы возможен ее запуск с места остановки. При этом программа продолжает выполняться корректно.

Отсутствие зависимости по данным делает процесс отладки программ на процессоре невозможным, так как для дальнейшего корректного выполнения программы необходимо восстановить точное состояние конвейера в момент остановки. Для этого процессор должен иметь специальные средства сохранения состояния конвейера. Если данная аппаратура отсутствует, то корректный запуск программы после остановки невозможен.

Таким образом, отладка программ для процессора, не имеющего аппаратной реализации механизма отслеживания зависимости по данным и специальной аппаратуры для сохранения состояния конвейера, возможна только на его потактовом симуляторе, что не всегда дает гарантии корректной отладки вследствие возможного несоответствия симулятора и реального процессора.

VI. Корректность программного обеспечения при модернизации конвейера

В процессе разработки и последующей модернизации процессора существуют ситуации, когда разработчику приходится изменять конвейер. Чаще всего конвейер удлиняют для уменьшения времени исполнения одного такта, соответственно, для повышения производительности процессора.

Для процессоров, реализующих цифровую обработку сигналов, разрабатывается стандартизированное программное обеспечение: библиотека стандартизированных функций.

Если процессор не имеет аппаратных средств отслеживания зависимостей по данным и ресурсам, то при модернизации конвейера в общем случае разработанное программное обеспечение становится некорректным и требует полной модификации.

Если же процессор имеет в своем составе аппаратуру для отслеживания зависимостей по данным и ресурсам, то программное обеспечение не требует модификации, библиотечные программы будут выполняться корректно, хотя при этом может снизиться эффективность выполнения. Для повышения эффективности исполнения программ, необходимо оптимизировать 20% кода, что, конечно, несравнимо с полной модификацией библиотеки.

VII. Заключение

В результате исследования **установлено**:

1) Отсутствие отслеживания зависимостей, в некоторых случаях позволяет повысить производительность программы на 12,5 %.
2) Отсутствие отслеживания зависимостей позволяет упростить архитектуру, что позволяет уменьшить необходимые аппаратные ресурсы и увеличить тактовую частоту примерно на 10%.
3) Введение аппаратного отслеживания зависимостей существенно упрощает разработку и отладку программного обеспечения.
4) Введение аппаратного отслеживания зависимостей позволяет сохранить совместимость ранее разработанного программного обеспечения в случае модернизации конвейера при развитии архитектуры.
5) Введение аппаратного отслеживания зависимостей позволяет реализацию прерываний и исключений.

По результатам исследований сделан вывод об оправданности реализации текущей версии микропроцессора КОМДИВ128-RIO без отслеживания зависимостей по данным, поскольку данное решение позволило сократить срок разработки микропроцессора и достичь заданных характеристик по тактовой частоте и потребляемой мощности.

При дальнейшем развитии архитектуры и разработке будущих версий микропроцессора КОМДИВ128-RIO рекомендуется реализовать аппаратное отслеживание зависимостей по данным, поскольку совместимость программного обеспечения, удобства отладки и возможность реализации прерываний и исключительных ситуаций будут иметь большее значение, нежели локальный выигрыш в технических характеристиках микропроцессора.

Литература

[1] Sima Deszo, “Decisive Aspects in the Evolution of Microprocessors. // Proc.IEEE. 2004. V. 92. N12. P. 1895-1926.
[2] G. Martin, S. Leibson. Processor Design Mistakes, Part 8: Ommiting Pipeline Interlocks Seemed Liked a Good Idea, Tencilica, Inc. 2008. www.mpr-online.ru.
[3] Микросхема интегральная МС-24R. Руководство пользователя. 2007. www.multicore.ru.
[4] Архитектура NeuroMatrix NM6403. Руководство пользователя. 2007. www.module.ru.
[5] TMS320C674x DSP CPU and Instruction Set, Texas Instruments, October 2008.
[6] Analog Devices, Inc. Data Sheet Final – ADSP-TS201 TigerSHARK Embedded Processor, Rev. C. December 2006. P. 3.
[7] ARM 11 MPCore Processor. Rev. r1p0. Technical Reference Manual. 2008. P. 19-7. ARM.