УДК 681.324

Программно-аппаратная библиотека математических функций для систем на ПЛИС

Зинченко Ю.Е., Гриценко А.А., Зеленева И.Я., Войтов Г.В.

Аннотация

Зинченко Ю.Е., Гриценко А.А., Зеленева И.Я., Войтов Г.В. Программно-аппаратная библиотека математических функций для систем на ПЛИС. *В статье предлагается метод создания библиотеки математических функций для систем на ПЛИС (SoPC) и набора необходимых для работы с ней аппаратно-программных средств. Предлагаемая библиотека дает возможность варьировать структуру системы на ПЛИС в соответствии с требованиями пользователя и доступными аппаратными ресурсами.*

Ключевые слова: ПЛИС, СИСТЕМА НА ПЛИС, ПРОЦЕССОР, ПЕРИФЕРИЙНЫЙ МОДУЛЬ, БИБЛИОТЕКА.

Анотація

Зінченко Ю.Є., Гриценко А.О., Зеленьова І.Я., Войтов Г.В. Програмно-апаратна бібліотека математичних функцій для систем на ПЛІС. *У статті пропонується метод створення бібліотеки математичних функцій для систем на ПЛІС (SoPC) і набору необхідних для роботи з нею апаратно-програмних засобів. Запропонована бібліотека надає можливість варіювати структурою системи на ПЛІС відповідно до вимог користувача та доступних апаратних ресурсів.*

Ключові слова: ПЛІС, СИСТЕМИ НА ПЛІС, ПРОЦЕСОР, ПЕРИФЕРІЙНИХ МОДУЛІВ, БІБЛІОТЕКА.

Abstract

Zinchenko Y.E., Grytsenko A.A., Zelenyova I.J., Voytov G.V. Hardware-software math library for SoPC. *In this paper is proposed a method of creation of the math library for SoPC and set of the required tools. Usage of this library allows effectively vary the structure of SoPC in concordance with user requirements depending on the available hardware resources.*

Keywords: FPGA, SOPC, CPU, PERIPHERAL MODULE, LIBRARY.

Введение

Внастоящее время производители ПЛИС предоставляют широкие возможности по созданию систем на ПЛИС с использованием различных микропроцессорных архитектур. В частности, речь идет об архитектурах Xilinx MicroBlaze [1] и Altera Nios II [2], которые

подразумевают наличие центрального микропроцессора и набора периферийных модулей, взаимодействующих посредством предоставляемого процессором шинного интерфейса.

Программное обеспечение рассматриваемых систем создается с использованием современных высокоуровневых языков программирования, в частности C и C++ [3, 4]. Оба эти языка предоставляют доступ к стандартной библиотеке математических функций. Необходимость использования функций математической библиотеки ведет к увеличению вычислительной нагрузки на центральный процессор.

Постановка задачи

Рассматриваемые в данной работе системы на ПЛИС, характеризуются наличием специфичных модулей, которые обеспечивают решение узкоспециализированных задач, позволяя центральному процессору переложить на них обязанности в области трудоемких вычислений (рис. 1), в частности, в области вычисления математических функций.

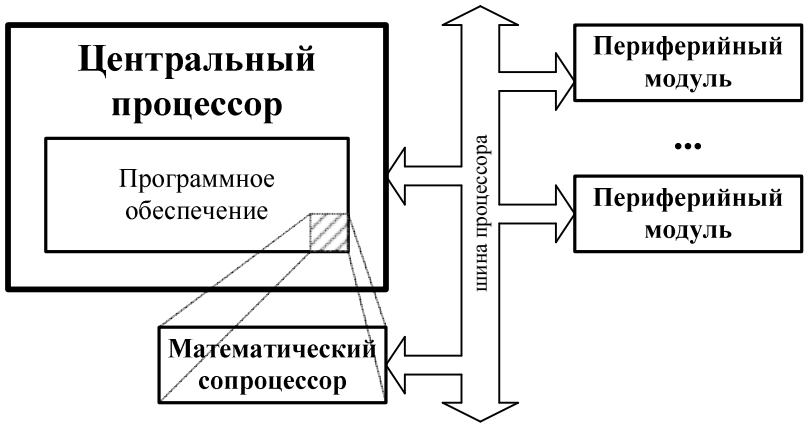

Рисунок 1 – Архитектура микропроцессорной системы на ПЛИС, включающей сопроцессор

Важным вопросом, который возникает в процессе создания таких программно-аппаратных систем, является организация взаимодействия сопроцессоров и центрального процессора. С точки зрения архитектуры это подразумевает создание дополнительного программного модуля, обеспечивающего доступ к каждому из сопроцессоров, а также, реализацию аппаратных модулей, используемых для подключения сопроцессоров к шине процессора. Для решения этого вопроса в данной работе предлагается подход, который позволяет изолировать логику математического сопроцессора от деталей его подключения к шине процессора, а логику программного обеспечения – от деталей взаимодействия с сопроцессором.

Использование аппаратного модуля для выполнения вычислений является, с одной стороны, производительным вариантом оптимизации, а с другой, ресурсоемким. Поэтому

реализация таких модулей требует использования подхода, который позволяет получить лучшие значения обеих характеристик в конкретном случае. Описание соответствующего подхода является вторым важным вопросом, который рассматривается в данной работе. Предлагаемое решение этого вопроса базируется на организации диалога с экспертом, обладающим знаниями предметной области реализуемого системой приложения. Целью такого диалога является выявление ограничений использования тех вычислительных компонент системы, которые будут реализованы в форме математических сопроцессоров.

Использование программно-аппаратной библиотеки и архитектура подключения сопроцессоров

Согласно [5] в процессе разработки программно-аппаратной системы пользователь не должен самостоятельно определять способ реализации отдельных компонентов. Для обеспечения работы пользователя требуется, во-первых, иметь некоторую формальную методику разработки систем на ПЛИС, обеспечивающую автоматизированный анализа ее структуры и поиск путей ее оптимизации. Во-вторых, обеспечить пользователя набором средств, необходимых для того, чтобы, с одной стороны, позволить ему управлять процессами формирования структуры системы, с другой – обеспечить качественную его изоляцию от деталей выполнения этих процессов.

Исходными данными будем считать некоторую систему на ПЛИС, использующую функции математической библиотеки. Здесь следует отметить необходимость того, чтобы разработчик использовал не стандартную математическую библиотеку, а специальный ее аналог, обеспечивающий дополнительную функциональность, которая необходима для дальнейшего анализа. В частности, речь идет о поддержке механизмов профилирования.

Первым этапом является предварительный анализ, который представлен на рис. 2 с использование нотации UML [6]. НА этом этапе выполняется получения результатов профилирования программной части. Эти результаты генерируются предоставляемым, специфичным для конкретной архитектуры, вариантом математической библиотеки.

Анализ результатов показывает основные возможности по оптимизации системы. Дальнейшие этапы могут повторяться циклически, для получения наиболее подходящего варианта, относительно повышения производительности и расхода аппаратных ресурсов.

Вторым этапом является получение экспертных знаний о том, каким образом должны или могут выполняться вычисления математических функций, которые были отобраны на первом этапе. Точность и полнота экспертных знаний влияет на результаты оптимизации системы. В качестве таких знаний могут выступать, например:

- *Способ представления чисел*. Если необходимо использовать представление с плавающей запятой, то, скорее всего, нужно будет использовать сторонние ядра для выполнения арифметических операций; если можно использовать представление с фиксированной запятой, то понадобятся соответствующие функции преобразования.
- *Необходимая точность результатов*. Допустим известно то, что требуемая точность не превышает некоторое количество разрядов после запятой. В этом случае можно сократить количество ресурсов, которые необходимы для размещения сопроцессора.
- *Допустимый диапазон входных значений*. Программная реализация часто отличается общностью, с другой стороны, аппаратная реализация позволяет, повысить производительность и уменьшить затраты ресурсов за счет своей специфичности.

Увеличение объема информации дает возможность создавать более специфичные, а, как следствие, и более производительные и менее затратные по ресурсам сопроцессоры.

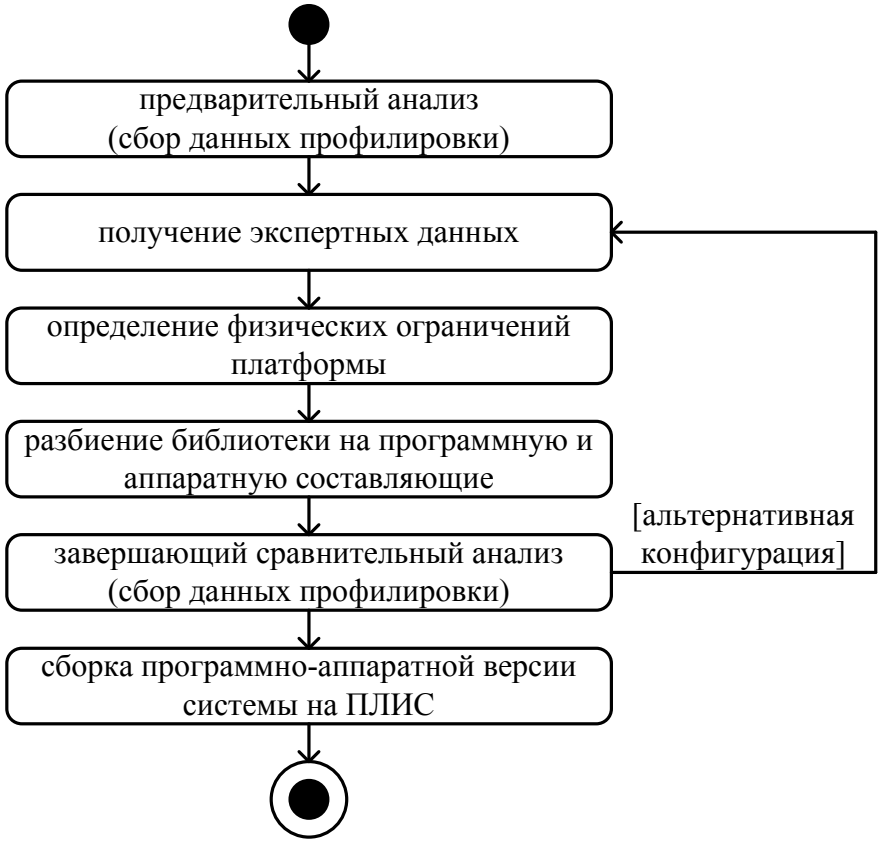

Рисунок 2 – Методика разработки системы на ПЛИС, использующей программно-аппаратную реализацию математической библиотеки

Следующим шагом является получение информации об используемой платформе. Эта информация позволяет выбрать наиболее подходящую для конкретной платформы реализацию аппаратного модуля, а также определить способ подключения модуля к шине процессора. Также появляется возможность определить механизмы доступа к модулю со стороны программного обеспечения.

Например, если известно, что будет использоваться ПЛИС FPGA Altera Cyclone II, то можно выбрать реализацию модуля, оптимизированную для архитектуры этой ПЛИС [7]. С другой стороны, подключение модуля к микропроцессору будет осуществляться через шину Avalon [8], что позволяет выбрать необходимый модуль сопряжения с шиной. Кроме того, можно однозначно определить способ взаимодействия с модулем со стороны программного обеспечения [3].

Предлагается использовать следующую архитектуру подключения сопроцессоров (рис. 3):

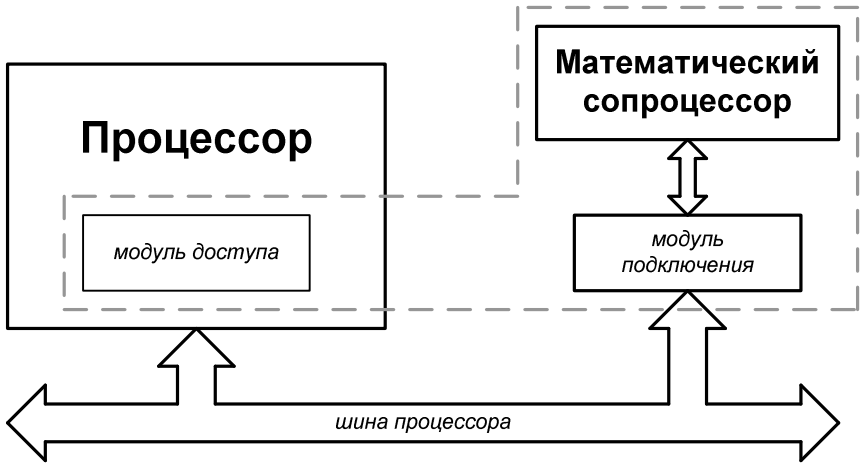

Рисунок 3 – Архитектура для подключения математического сопроцессора

В предлагаемой архитектуре выделен модуль подключения. Использование этого модуля позволяет использовать для нескольких платформ один неоптимизированный вариант сопроцессора. Этот модуль должен обеспечивать корректный протокол работы с сопроцессором, например, в случае использования интерфейса отображения в памяти шины Avalon [8], он обеспечивает запись и чтение данных по соответствующим адресам.

Следующим этапом разработки является разбиение математической библиотеки на программную и аппаратную часть. На этом этапе возникает необходимость в получении от пользователя информации о том, сколько и каких ресурсов ПЛИС можно использовать. К этому моменту точно известно, сколько ресурсов необходимо для размещения того или

иного сопроцессора, поэтому, в зависимости от имеющихся ресурсов можно определить все допустимые конфигурации системы.

Последним этапом является сравнительный анализ. На этом этапе используется профилирующая версия математической библиотеки, что позволяет оценить полученный в результате оптимизации прирост производительности. По необходимости разработчик получает возможность сравнить несколько альтернативных конфигураций системы.

Структура программно-аппаратной библиотеки

Следует отметить, что с точки зрения системы на ПЛИС, структура библиотеки в процессе разработки изменяется. При использовании языков C и C++, библиотека будет состоять из изначально двух, а позже из трех, файлов. Заголовочный файл обеспечивает возможность взаимодействия с библиотекой, а два файла исходных кодов (или два объектных файла) содержат реализацию программной составляющей и модулей доступа к сопроцессорам (рис. 3). Аппаратная часть библиотеки зависит от того, какие функции и, в какой конфигурации, погружены в аппаратуру.

В процессе разработки начальной версии системы и на этапе предварительного анализа программно-аппаратная библиотека является оберткой стандартной библиотеки и имеет полностью программную реализацию. Ее основным отличием является то, что она позволяет осуществить получение результатов профилирования (рис. 4).

По завершении сбора экспертной информации и данных об используемой платформе выполняется генерация специализированной версии библиотеки, которая, все же, позволяет осуществить профилирование (рис. 5).

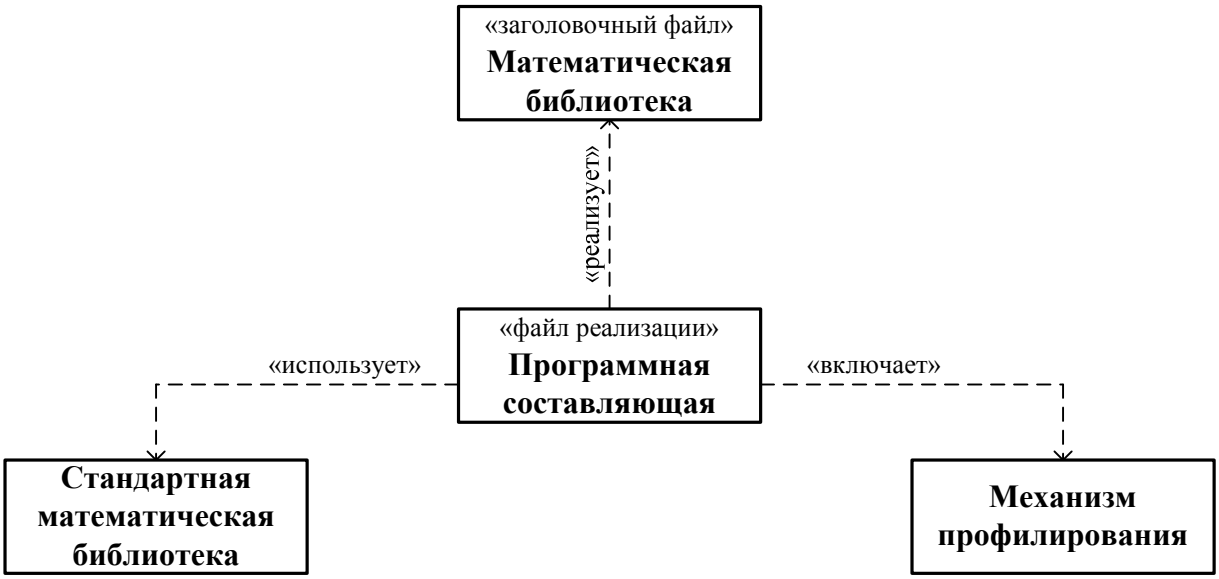

Рисунок 4 – Первоначальная структура библиотеки

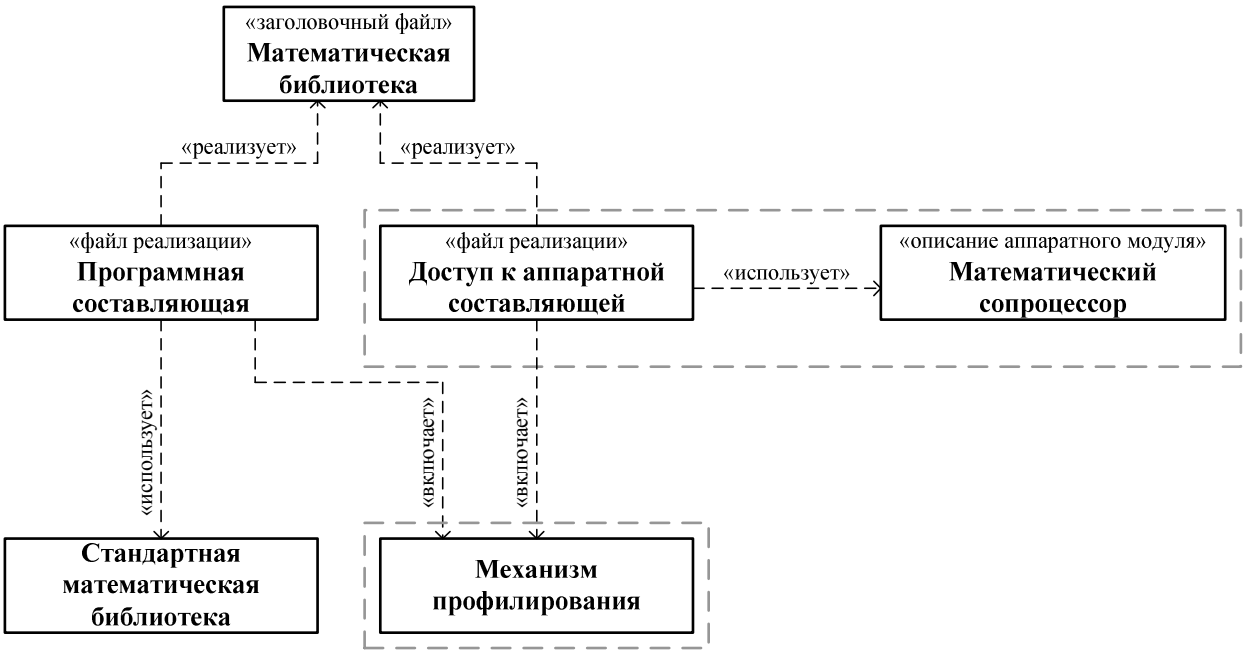

Рисунок 5 – Окончательный вариант структуры библиотеки

Это нужно для выполнения сравнительного анализа исходной и оптимизированной версий, а также, в случае необходимости, нескольких разных оптимизированных версий.

Механизмы профилирования удаляются из библиотеки только после того, как будет выбрана подходящая пользователю конфигурация.

Пример реализации экспоненциальной функции

Допустим, что согласно результатам профилирования программного обеспечения для системы на ПЛИС, значительная часть вычислительных ресурсов расходуется на вычисление экспоненциальной функции [9]. Согласно ранее предложенной методики, следующим этапом является получение ограничений пользователя. Пусть, пользователем задан диапазон для входных значений от нуля до двух, включительно, точность в пять знаков после запятой и выбран формат с фиксированной запятой. В качестве платформы была выбрана ПЛИС FPGA Xilinx Spartan3AN.

На основе приведенной экспертной информации, можно предложить следующий вариант реализации модуля доступа (рис. 6):

Следует обратить внимание на использование механизмов утверждений [3, 4]. Они позволяют, с одной стороны, контролировать экспертные утверждения об использовании функции, с другой, вхождение полученных результатов в допустимый диапазон. Кроме того,

показанный исходный код содержит дополнительную проверку, гарантирующую сравнение результатов работы сопроцессора и ожидаемых результатов с учетом заданной точности.

```
double hs_math_exp (double x) {
     assert (0.0 <= x && x <= 2.0);                          } Шаг 1. Контроль и подготовка исходных данных
     Xuint32 x_fixed = to_fixed (x);

     XIo_Out32 (HS_MATH_EXP_NPU, x_fixed);                   } Шаг 2. Расчет с использованием сопроцессора
     Xuint32 y_fixed = XIo_In32 (HS_MATH_EXP_NPU);

     double y = from_fixed (y_fixed);                        } Шаг 3. Контроль, подготовка и выдача результатов
     assert (1.0 <= y && y <= 7.38906);
     assert (are_equal (y, exp (x), 5));
     return y;
}
```

Рисунок 6 – Пример реализации специализированного модуля доступа к сопроцессору, вычисляющему экспоненциальную функцию (для архитектуры Xilinx MicroBlaze [1])

Результаты реализации экспоненциальной функции

Даже в случае использования неоптимизированной реализации сопроцессора, время работы варианта из стандартной библиотеки превышает время работы аппаратного модуля в 90 раз, в лучшем случае (рис. 7), что доказывает эффективность предложенного подхода.

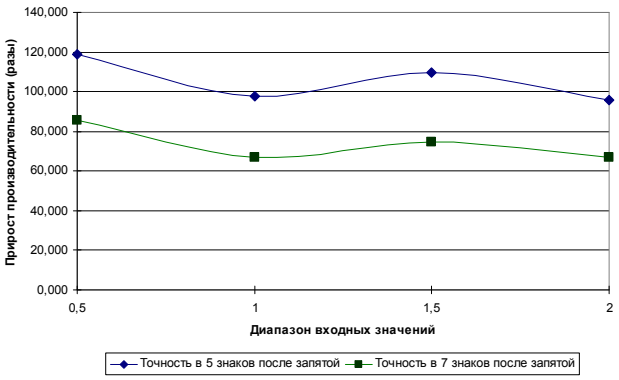

Рисунок 7 – Прирост производительности при использовании аппаратного модуля

В качестве примера, который иллюстрирует важность качественной обработки экспертных данных, на диаграмме (рис. 8) показан прирост для точности в семь знаков. Следует отметить, что прирост аппаратных затрат в этом случае составил порядка 25% относительно модуля с точностью в пять знаков.

Заключение

В данной статье предложен метод создания библиотеки математических функций для систем на ПЛИС. Данный метод позволяет выполнять качественную оптимизацию систем на ПЛИС, которая основывается на двух важных факторах – экспертных знаниях о предметной области приложения и информации об используемой платформе.

Применение предложенного метода позволяет повысить производительность, что доказывают полученные результаты экспериментов.

Разработка библиотеки, обеспечивающей поддержку перечисленных платформ, сейчас ведется на базе лаборатории «FPGA-технологий и диагностики КС» ДонНТУ.

Список источников

1. MicroBlaze Processor Reference Guide [Electronic Resource] – Mode of access URL http://www.xilinx.com/support/documentation/sw_manuals/edk63i_mb_ref_guide.pdf – Title from the screen.
2. Nios II Processor Reference Handbook [Electronic Resource]. – Mode of access : URL http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf – Title from the screen.
3. Programming languages – C: ISO/IEC 9899:1999(E) – ISO/IEC 9899/1999(E) – [Чинний від 01.12.1999] – ISO/IEC, 1999. – 538 p. – International Standard.
4. Programming languages – C++: ISO/IEC 14882:2003 – ISO/IEC 14882:2003 – [Чинний від 01.12.2003] – ISO/IEC, 2003. – 650 p. – International Standard.
5. Jerraya A.A. Hardware/Software Interface Codesign for Embedded Systems / A.A. Jerraya, W. Wolf // Computer – 2005 – № 38 – P. 63-69
6. Объектно-ориентированный анализ и проектирование с примерами приложений / [Г. Буч, Р. Максимчук, М. Энгл и др.] – [3-е изд.] – М. : ООО "И.Д. Вильямс", 2008. – 720 с.: ил.
7. Cyclone II Device Handbook Volume 1 [Electronic Resource] – Mode of access : URL http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf – Title from the screen.
8. Avalon Interface Specifications [Electronic Resource] – Mode of access : URL http://www.altera.com/literature/manual/mnl_avalon_spec.pdf – Title from the screen.
9. Ильин В.А. Математический анализ. Начальный курс / Ильин В.А., Садовничий В.А., Сендович Б.Ч. – [2-е изд., перераб.] – М.: Изд-ва МГУ, 1985. – 662 с.