НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ноябрь–декабрь 2022 Том 22 № 6 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
November–December 2022 Vol. 22 No 6 http://ntv.ifmo.ru/en/
ISSN 2226-1494 (print) ISSN 2500-0373 (online)

doi: 10.17586/2226-1494-2022-22-6-1136-1142
УДК 004.2, 004.3, 004.383.8, 004.383.8.032.26, 004.4, 004.273

Метод документирования архитектурных решений вычислительных платформ

Ярослав Георгиевич Горбачев[1,2]✉

[1] ООО ЛМТ, Санкт-Петербург, 199034, Российская Федерация
[2] Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация
yaroslav-go@yandex.ru✉, https://orcid.org/0000-0001-5419-6422

Аннотация

Предмет исследования. Представлен метод описания вычислительных механизмов и документирования вычислительных платформ. Новизна метода заключается в использовании унифицированных средств для документирования на разных уровнях гранулярности аппаратного, программного и инструментального обеспечения, а также реконфигурируемых (интеллектуальных, адаптивных) компонент. Метод позволяет представлять в понятном виде вычислительные системы с нестандартными архитектурными решениями. **Метод.** Предложенный метод заключается в описании идеальной модели вычислительной платформы с ее последующей итеративной детализацией. Особенность метода — наличие единого для различных систем ядра, которое используется для упрощения описаний и структурирования информации. Ядро включает в себя универсальные элементы и создано на основе анализа большого количества архитектур вычислительных систем. **Основные результаты.** С использованием предложенного метода описаны принципы организации широкого спектра вычислительных платформ. Рассмотрены следующие платформы: обобщенные процессоры с классической архитектурой, которая является развитием принципов фон-Неймана; программно-аппаратные системы на базе микроконтроллеров; операционные системы общего назначения; крупногранулярные и мелкогранулярные реконфигурируемые вычислительные системы; специализированные процессоры и ускорители; искусственные нейронные сети. **Практическая значимость.** Предложенный метод может быть использован для структурирования информации как по традиционным, так и по активно развивающимся направлениям: реконфигурируемым вычислительным системам и специализированным процессорам. На основе метода создана общая база универсальных вычислительных механизмов, пригодных для использования в разных узлах системы, для объектов разной гранулярности программными, аппаратными и иными средствами, на разной элементной базе. Результаты работы могут быть полезны системным архитекторам для документирования сложных вычислительных компонент, состоящих из программных, аппаратных и прочих механизмов. Метод направлен на упрощение повторного использования вычислительных механизмов и призван облегчить генерацию новых архитектурных решений. Также метод может быть полезен при обучении профильных специалистов, поскольку позволяет демонстрировать основные принципы вычислительной техники.

Ключевые слова

архитектура вычислительной системы, архитектурное описание, вычислительная платформа, вычислительный механизм, реконфигурируемая система, динамическая реконфигурация

Ссылка для цитирования: Горбачев Я.Г. Метод документирования архитектурных решений вычислительных платформ // Научно-технический вестник информационных технологий, механики и оптики. 2022. Т. 22, № 6. С. 1136–1142. doi: 10.17586/2226-1494-2022-22-6-1136-1142

A method for documenting architectural solutions of computing platforms

Yaroslav G. Gorbachev[1,2]✉

[1] LMT Ltd., Saint Petersburg, 199034, Russian Federation
[2] ITMO University, Saint Petersburg, 197101, Russian Federation
yaroslav-go@yandex.ru✉, https://orcid.org/0000-0001-5419-6422

Abstract
The article describes a method for documenting the principles of functioning and internal organization of computing platforms, including reconfigurable computing systems and non-standard processor architectures. The novelty is in using of unified tools to describe: the design process and the computing process, hardware, software and tools, computing components of different granularity. The proposed approach is to describe computing platform as an ideal model that represents abstract algorithms for fulfilling functional requirements without specifying of how to implement it. Then the iterative model refinement follows including selection of physical implementation options, specifying the technological stacks, and additional mechanisms that provide the specified system qualities. A feature of the method is a kernel used for structuring information, classifying and describing the computational mechanisms the system consists of. The kernel includes elements common for different systems and is based on the analysis of a large number of computing architectures. The method describes the principles of the organization of platforms which are usually not considered together. These are: generalized processors with classical architecture which is an evolution of the von Neumann principles; systems based on microcontrollers; operating systems; large- and small-granular reconfigurable systems; specialized processors and accelerators; artificial neural networks. The proposed method can be used to structure information in both traditional and rapidly developing areas: reconfigurable systems and specialized processors. Based on the method, it is possible to create a common database of computing mechanisms suitable for use in different functional units of the system and at different levels of granularity. The results of the work can be useful for system architects to describe complex computing mechanisms consisting of software, hardware and dynamically generated adaptive "intelligent" components which will simplify their reuse and can be used to generate new architectural solutions. Also, the proposed method can be used in the process of training specialists, for a visual demonstration of the basic principles of computer technology.

Keywords
architecture, architectural description, computing systems, computing mechanisms, reconfigurable systems

For citation: Gorbachev Ya.G. A method for documenting architectural solutions of computing platforms. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2022, vol. 22, no. 6, pp. 1136–1142 (in Russian). doi: 10.17586/2226-1494-2022-22-6-1136-1142

Введение

Современная компьютерная индустрия сложна и разделена на самостоятельные, обособленно развивающиеся направления (например, разработка программного и аппаратного обеспечения). Как следствие, многие специалисты компетентны лишь в какой-то узкой области, плохо представляя, что происходит за ее границами.

Семантическая и технологическая разобщенность специалистов продолжает стремительно расти с развитием реконфигурируемых систем [1] и обилием оригинальных архитектур и концепций, например: NISC-систем [2], вычислений с памятью [3] и в памяти [4], процессоров функционального уровня [5], WARP-процессоров [6] и др. Каждое из нестандартных архитектурных решений требует уникального инструментария и компетенций разработчиков, обладает собственным набором характеристик (производительность, надежность, стоимость, сложность и т. д.).

Необходимость изучения нестандартных архитектур вычислительных систем связана с окончанием действия «Закона Мура» и поиском новых путей повышения эффективности вычислений, в том числе за счет преодоления проблемы «бутылочного горлышка» классических архитектур [4].

Одна из проблем — обмен информацией между сотрудниками об использованных в проектах архитектурных решениях и перенос смешанных программно-аппаратных решений между разными вычислительными платформами [7]. В этом случае общепринятое разделение на программное обеспечение (ПО) и аппаратуру неэффективно и может снизить производительность команды разработчиков.

В некоторых программно-аппаратных решениях отдельные действия, обычно выполняемые во время разработки, смещаются в область функционирования (или наоборот). Классификации реконфигурируемых систем и систем, реализующих параллельные вычисления, часто оперируют понятием «гранулярность», которое может относиться как к разбиению алгоритмов на составляющие, так и к данным.

В связи с этим необходима разработка способа регистрации наиболее значимой информации о рассматриваемых архитектурах и вычислительных механизмах без искусственного разделения на программное, аппаратное и инструментальное обеспечения, на этапы проектирования и непосредственного исполнения для объектов разной гранулярности.

Если говорить об известных решениях, позволяющих описать принципы функционирования вычислительных систем, в первую очередь следует упомянуть понятие «модели вычислений» [8]. Модели вычислений определяют формальные правила и ограничения, описывающие поведение вычислительной системы или ее виртуального представления.

Для описания аппаратуры вычислительных систем на архитектурном уровне существует целый набор языков описания архитектуры [9]. К сожалению, они в основном узко специализированы и имеют значительные ограничения.

Довольно универсальные архитектурные описания доступны через создание высокоуровневых моделей в рамках методологии модельно-ориентированного проектирования [10]. Данная методология поддерживается рядом широко используемых в индустрии инструментов, таких как Simulink и LabVIEW. Отдельным направлением является генерация аппаратуры с помощью средств HLS [11].

Распространено описание ПО и вычислительных систем на основе языка UML [12] и его подмножества

SysML [13]. Эти языки широко поддерживаются разнообразными инструментальными средствами, например средой моделирования Capella и методом ARCADIA [14].

В слабо формализованной области программной инженерии, где разработчики столкнулись со схожими проблемами (большое количество различных сущностей, которые сложно классифицировать, сравнивать и описывать), используется стандарт Essence[1], созданный для описания и применения на практике методов программной инженерии.

Ни один из описанных подходов в полной мере не удовлетворяет заданным требованиям.

Описание модели

Отправной точкой для описания специализированных вычислительных архитектур и вычислительных механизмов предложено взять некую идеальную модель, аналог абсолютно черного тела и других подобных физических моделей. Модель должна быть пригодна для описания как можно большего количества разнообразных вычислительных механизмов и платформ, а также отображать наиболее значимые аспекты функционирования и жизненного цикла при максимальной простоте для восприятия. Сформулируем определение вычислительной платформы как набора программных и аппаратных средств (или стандарта их организации) для реализации прикладных вычислений, а вычислительного механизма — как упрощенного принципа функционирования и организации отдельного аспекта или элемента вычислительной платформы.

Для создания модели наиболее перспективен подход, предложенный в стандарте Essence. Подобно выделению общего ядра из универсальных элементов для методов и практик программной инженерии в Essence, в области вычислительной техники также можно выделить общее ядро — основу для описания широкого спектра вычислительных систем и механизмов. Ядро Essence включает в себя универсальные элементы и не зависит от конкретных реализаций и их практических применений. Отдельные элементы выделены из большого количества конкретных примеров, а все реализации описаны как надстройки поверх неизменного ядра.

Один из простейших способов описания объекта — его представление в виде «черного ящика», когда описывается взаимодействие объекта с внешним миром и реакции на внешние раздражители, а внутреннее устройство не раскрывается. Такой «черный ящик» принят за основу предложенной модели.

Система, рассматриваемая как «черный ящик», каким-либо образом производит преобразование подаваемой на вход энергии (соответствующие уровни которой могут интерпретироваться как данные) из некоторых входных значений в соответствующие им выходные. Кроме этого, в общем случае вычислительные (и не только) системы могут иметь возможность сохранять состояние.

Для системы как «черного ящика» можно выделить следующие обобщенные действия или этапы функционирования, которые составляют основу ядра:

1) чтение входов;
2) вычисление;
3) сохранение внутреннего состояния;
4) запись выходов.

Отметим, что любая система обязательно должна выполнять хотя бы часть перечисленных действий. Описанное ядро — основа идеальной модели системы, на которую надстраивается ее более подробное описание.

Так как модель идеальная, считаем, что все действия производятся мгновенно, объемы блоков памятей бесконечны, и не требуется дополнительных действий, например обеспечения системы энергией, необходимой для штатного функционирования.

Примем, что переход от действия к действию идет сверху вниз с возможными возвратами назад и повторением отдельных итераций столько раз, сколько необходимо. Отметим, что в отдельных случаях может быть другой порядок действий.

При рассмотрении вычислителя в контексте жизненного цикла, необходимо учесть такие действия, как создание и уничтожение объекта. Контекст вычислительной техники делает оправданным выделение следующих этапов в создании объекта: создание реально существующей в физическом мире и фиксированной основы (аппаратного обеспечения) и ее программирование. В данном случае программирование необходимо понимать как программирование в широком смысле, включая создание и последующую запись в целевое устройство программ, настроек, конфигураций программируемой логики и т. д.

Отметим отличие между созданием физической основы и ее программированием. Создание физической основы производится только один раз, при создании системы, и закладывает неизменные на всех этапах жизненного цикла принципы (если оставить за скобками деградацию компонентов, помехи и т. д.). Программирование может (но не обязательно должно) производиться больше одного раза, явно и заметно изменяя свойства системы.

Обобщенно можно выделить следующие этапы жизненного цикла некоторой системы, подсистемы или элемента ядра. При необходимости они могут варьироваться и дополняться (приведенный список — лишь один из возможных вариантов представления):

— фиксация основных принципов функционирования системы;
— создание инструментария для разработки аппаратуры;
— разработка аппаратуры;
— создание инструментария для разработки ПО;
— разработка ПО;
— создание инструментария для загрузки ПО;
— загрузка ПО;
— инициализация системы;
— штатное функционирование;
— уничтожение.

[1] Essence — Kernel and Language for Software Engineering Methods (Essence), v1.2., 300 с. [Электронный ресурс]. Режим доступа: https://www.omg.org/spec/Essence/1.2/PDF, свободный. Яз. англ. (дата обращения: 13.11.2022).

Описанная модель применима для простых систем, или для небольших подсистем, входящих в сложные системы. Для поверхностного понимания архитектуры и особенностей функционирования сложной системы в целом ее необходимо представлять на более детальном уровне. Таким уровнем может стать представление системы как взаимодействующих «черных ящиков» — атомарных вычислителей, каждый из которых осуществляет какое-то простейшее действие в составе системы.

Тем самым можно получить естественную плавающую границу, позволяющую повышать или понижать уровень абстракции. Сначала в самом грубом приближении мы рассматриваем систему как «черный ящик»; когда этого недостаточно — как систему взаимодействующих «черных ящиков»; когда и этого недостаточно — каждый из «черных ящиков», составляющих систему, рассматриваем как систему взаимодействующих «черных ящиков», и так до тех пор, пока не достигается нужный уровень детализации.

Систему «черных ящиков» уже не описать простыми действиями 1–4, рассредоточенными по этапам жизненного цикла. Результаты вычислений системы как целого есть производная от результатов вычислений каждого из составляющих систему «черных ящиков» и зависят от того, как они взаимодействуют.

При переходе от представления системы как «черного ящика» к представлению системы как взаимодействующих «черных ящиков» появляется необходимость описать устройство обеспечивающей среды — то, каким образом вычислительные блоки, блоки памяти, входы и выходы объединяются в единое целое и благодаря чему они получают возможность функционировать вместе, решая некоторые общие задачи.

Обеспечивающая среда идеальной модели должна выполнять две функции: перемещать данные между вычислительными блоками, блоками памятей, входами и выходами, и синхронизировать вычисления отдельных вычислительных блоков.

На этом этапе логично выделить управления данными и вычислениями. При этом область управления вычислениями не включает в себя сами вычисления — она только выявляет причинно-следственные связи и обеспечивает передачу управления конкретным вычислителям, «черным ящикам», в которых и производятся вычисления.

Для получения описания модели всей системы необходимо описать взаимодействие трех областей: «черных ящиков» или «атомарных» вычислений; подсистемы управления вычислителями; подсистемы управления данными. Вместе данные области формируют ядро идеальной модели системы при рассмотрении ее как системы «черных ящиков». Каждое действие, которое формирует ядро вычислительной системы, должно обеспечиваться цепочкой действий и этапов жизненного цикла.

Получение информации о желаемом поведении системы и ее «обучении» (например, анализ функциональных требований, написание удовлетворяющей им программы и ее прошивка), могут выполняться на этапах создания системы или в процессе функционирования для самообучающихся систем. В последнем случае обучение можно добавить в расширенный вариант ядра для каждой из подсистем, иначе обучение в ядро не включается и входит в процесс проектирования.

Подсистема управления вычислениями включает следующие этапы функционирования (вариант без возможности самообучения): определение следующего вычисления по предыдущему; запуск вычисления.

Подсистемы управления данными включает следующие этапы функционирования: чтение входов системы, блоков памятей, выходов вычислителей; коммутация данных внутри системы; запись данных в вычислители, блоки памяти и выходы системы.

Самообучение может быть добавлено в любую из двух подсистем и в сам вычислитель. Самообучение включает в себя получение и загрузку (реализацию) правил функционирования.

Представленная модель может быть использована для описания механизмов разной гранулярности. В качестве градаций гранулярности для вычислений можно выделить: инструкцию, базовый блок, функцию и процесс. Гранулярность для данных включает: бит данных, любое количество бит данных (байт, слово), пакет данных, файл и др. При необходимости можно наряду с цифровыми рассматривать аналоговые сигналы.

Примеры использования модели

Корректность модели была проверена на следующих примерах: основные архитектурные решения, используемые в типичном современном процессоре; программа для микроконтроллера; программируемая логическая интегральная схема (FPGA); операционная система; процессор функционального уровня (Function-Level Processor); математическая модель перцептрона. Была рассмотрена классификация вычислительных механизмов, обеспечивающих надежную работу подсистемы памяти с разнотипными блоками в контроллерах для ответственных применений.

Приведем пример описания архитектуры типичного современного процессора (рисунок), в котором использованы принципы фон-Неймана и стандартные архитектурные решения. Пример выбран как наиболее универсальный и понятный широкому кругу специалистов, метод может быть использован для описания более специфичных и нестандартных вычислительных архитектур.

На схеме показаны только наиболее важные моменты, дающие базовое понимание устройства процессора — основы, вынесенные на верхний уровень описания. При необходимости более детального описания любой из элементов, представленный на схеме (как этап функционирования, так и этап жизненного цикла) может быть расписан более подробно (на отдельной схеме или в «теле» основной схемы). Аналогичным образом описаны все вспомогательные механизмы — контроллер прерываний, механизмы работы со стеком, кэш, конвейер процессора, средства распараллеливания на уровне инструкций и др.

На «верхний уровень» описания вынесены следующие принципы и механизмы:

— универсализация и разделение ресурса во времени — использование одного универсального

функционального блока, арифметико-логического устройства (АЛУ) процессора для выполнения различных операций;
— использование шины данных под управлением отдельно добавленных в программный код инструкций — простое базовое действие, «считать данные со входов системы/из блоков памяти/с выходов вычислителей», разложенное в цепочку действий, выполняемых сначала программистом при написании программы, а затем процессором в процессе ее исполнения;
— формирование последовательности выполнения операций с помощью механизмов управления программным счетчиком и написанного вручную программистом алгоритма, хранящегося в памяти.

На рисунке представлен единый алгоритм функционирования и создания системы с разделением на три этапа жизненного цикла: функционирование; создание аппаратного обеспечения; создание ПО (при необходимости количество этапов можно увеличить, добавив, например: изготовление системы, загрузку ПО, инициализацию и т. д.). Горизонтально ориентированные пятиугольники в области, обозначенной как «RUNTIME PROCESS», описывают вычислительный процесс — действия, производимые системой на этапе функционирования. Вертикально ориентированные пятиугольники в областях, обозначенных как «HARDWARE DESIGN» и «SOFTWARE DESIGN», описывают действия, производимые на этапах создания аппаратуры и ПО соответственно, без которых само функционирование системы невозможно.

В соответствии с предложенным устройством ядра, действия этапа функционирования разделены на три подсистемы: вычисления (Computations); подсистема управления вычислениями (Computation management subsystem, которой в процессоре соответствует блок, обычно называемый «Countrol Unit», CU); подсистема управления данными (Data management subsystem). Отметим, что каждое действие этапа функционирования относится к одному из базовых действий ядра, обозначенных небольшими прямоугольниками, в которые они вписываются. Например, для подсистемы управления вычислениями это: определение следующего вычисления по предыдущему (Select next computation), которое заключается в чтении регистра счетчика команд (PC) и запуск вычисления (Launch next computation), который подразумевает под собой загрузку соответствующей команды в регистр команд (IR).

В описанном примере обучение вычислительного блока системы — АЛУ (ALU) процессора — происходит на этапах создания аппаратуры и разработки архитектуры команд (ISA). На рисунке этот этап обозначен как TRAINING (ISA design). Последовательность выполнения отдельных операций с помощью АЛУ и коммутация данных внутри системы формируются на этапе создания ПО и обозначены как «Writing Program». На схеме два элемента с таким названием, поскольку в первом случае подразумевается исключительно формирование последовательности вычислений, а во втором — добавление в программу отдельных инструкций, обеспечивающих взаимодействие с памятью. Тем не менее обе эти работы выполняются программистом

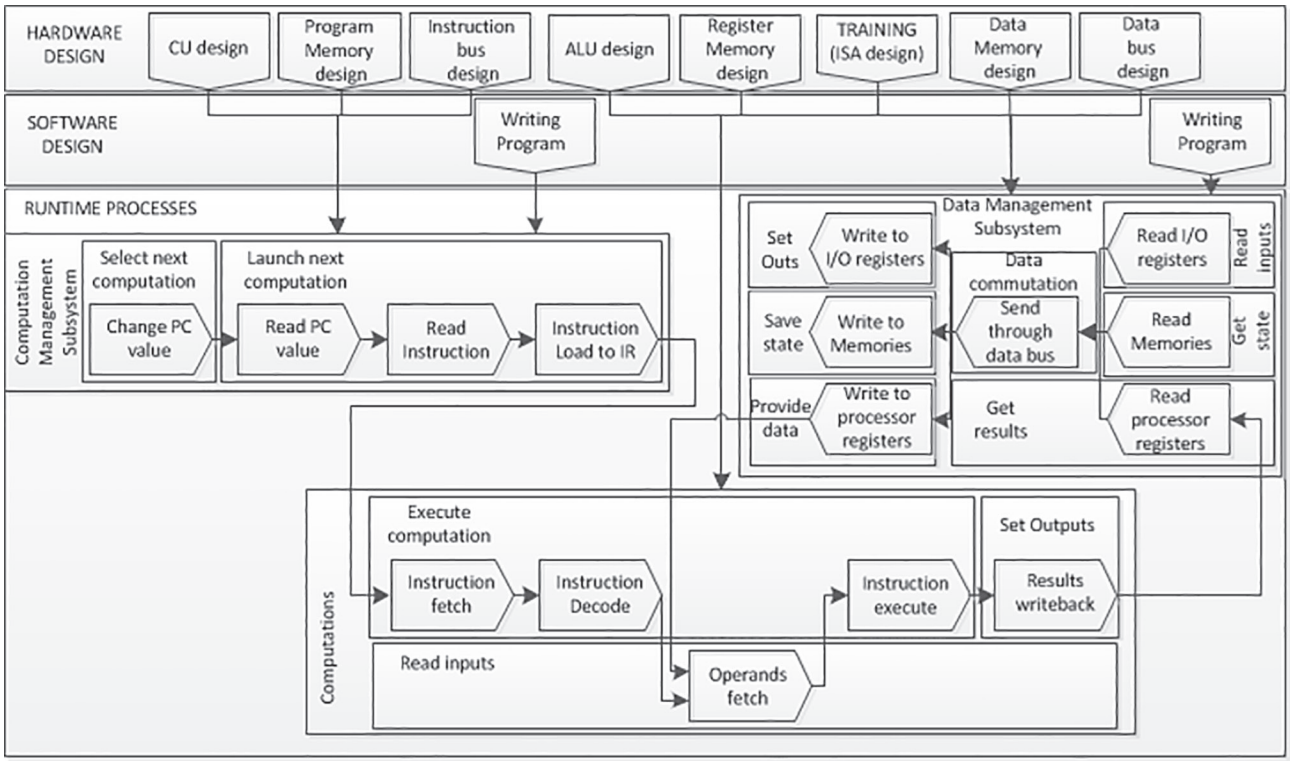

Рисунок. Пример документирования основных архитектурных решений типичного современного микропроцессора
Figure. An example of documenting the basic architectural solutions of a typical modern microprocessor

при написании ПО и обычно не разделяются на самостоятельные сущности.

Заключение

В работе предложен метод документирования архитектурных решений вычислительных платформ безотносительно к способу их конечной реализации, когда в единых терминах описываются: процесс проектирования и вычислительный процесс, аппаратура, программное обеспечение и инструментарий. Плавающий уровень абстракции позволяет получить описания для объектов разной гранулярности вычислений и данных.

Метод предоставляет особые правила структурирования информации об организации вычислительных платформ и о протекающих в них процессах. Основная цель метода — борьба со сложностью и упрощение обмена информацией с одновременным увеличением ее объема (рассматривается одновременно программное обеспечение, аппаратура и инструментальные аспекты создания системы).

Использование метода дает возможность взглянуть по-новому на многие вопросы и позволяет генерировать новые нестандартные решения. Одно из главных достоинств метода — наглядность представления протекающих в системе процессов и возможность выделения основной сути функционирования системы — образующих ее вычислительных механизмов, для их отдельного описания и дальнейшего повторного использования.

Представленный метод в первую очередь предназначен для системных архитекторов и технических писателей, но может быть использован в процессе обучения специалистов соответствующих направлений. В частности, метод прошел апробацию в Университете ИТМО в рамках магистерской программы «Компьютерные системы и технологии».

Литература

1. Chattopadhyay A. Ingredients of adaptability: A survey of reconfigurable processors // VLSI Design. 2013. P. 1–18. https://doi.org/10.1155/2013/683615
2. Reshadi M. No-Instruction-Set-Computer (NISC) Technology Modeling and Compilation: PhD dissertation / University of California, Irvine. 2007. 153 p.
3. Somnath P., Swarup B. Computing with Memory for Energy-Efficient Robust Systems. Dordrecht: Springer, 2011. 249 p.
4. Siegl P., Buchty R., Berekovic M. Data-centric computing frontiers: A survey on processing-in-memory // Proc. of the Second International Symposium on Memory Systems (MEMSYS ‘16). 2016. P. 295–308. https://doi.org/10.1145/2989081.2989087
5. Tabkhi H., Bushey R., Schirner G. Function-level processor (FLP): A novel processor class for efficient processing of streaming applications // Journal of Signal Processing and Systems. 2016. V. 85. N 1. P. 287–306. https://doi.org/10.1007/s11265-015-1058-5
6. Lysecky R., Stitt G., Vahid F. Warp Processors // ACM Transactions on Design Automation of Electronic Systems. 2006. V. 11. N 3. P. 659–681. https://doi.org/10.1145/1142980.1142986
7. Pinkevich V., Platunov A., Gorbachev Y. Design of embedded and cyber-physical systems using a cross-level microarchitectural pattern of the computational process organization // CEUR Workshop Proceedings. 2020. V. 2893.
8. Savage J. Models of Computation: Exploring the Power of Computing. Boston, MA, USA: Addison-Wesley, 1998. 600 p.
9. Processor Description Languages / ed. by M. Prabhat, D.Nikil. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2008. 432 p. https://doi.org/10.1016/B978-0-12-374287-2.X5001-0
10. Aarenstrup R. Managing Model-Based Design. Natick, MA: MathWorks Inc., 2015. 86 p.
11. Nane R., Sima V., Pilato C., Choi J., Fort B., Canis A., Chen Y., Hsiao H., Brown S., Ferrandi F., Anderson J., Bertels K. A Survey and evaluation of FPGA high-level synthesis tools // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2016. V. 35. N 10. P. 1591–1604. https://doi.org/10.1109/TCAD.2015.2513673
12. Booch G., Jacobson I., Rumbaugh J. The Unified Modeling Language User Guide. Boston, MA, USA: Addison-Wesley, 1998. 391 p.
13. Delligatti L. SysML Distilled: A Brief Guide to the Systems Modeling Language. Boston, MA, USA: Addison-Wesley, 2013. 267 p.
14. Jean-Luc V. Model-based System and Architecture Engineering with the Arcadia Method. Elsevier, 2017. 388 p.

References

1. Chattopadhyay A. Ingredients of adaptability: A survey of reconfigurable processors. *VLSI Design*, 2013, pp. 1–18. https://doi.org/10.1155/2013/683615
2. Reshadi M. *No-Instruction-Set-Computer (NISC) Technology Modeling and Compilation*. PhD dissertation. University of California, Irvine. 2007, 153 p.
3. Somnath P., Swarup B. *Computing with Memory for Energy-Efficient Robust Systems*. Dordrecht, Springer, 2011, 249 p.
4. Siegl P., Buchty R., Berekovic M. Data-centric computing frontiers: A survey on processing-in-memory. *Proc. of the Second International Symposium on Memory Systems (MEMSYS ‘16)*, 2016, pp. 295–308. https://doi.org/10.1145/2989081.2989087
5. Tabkhi H., Bushey R., Schirner G. Function-level processor (FLP): A novel processor class for efficient processing of streaming applications. *Journal of Signal Processing and Systems*, 2016, vol. 85, no. 1, pp. 287–306. https://doi.org/10.1007/s11265-015-1058-5
6. Lysecky R., Stitt G., Vahid F. Warp Processors. *ACM Transactions on Design Automation of Electronic Systems*, 2006, vol. 11, no. 3, pp. 659–681. https://doi.org/10.1145/1142980.1142986
7. Pinkevich V., Platunov A., Gorbachev Y. Design of embedded and cyber-physical systems using a cross-level microarchitectural pattern of the computational process organization. *CEUR Workshop Proceedings*, 2020, vol. 2893.
8. Savage J. *Models of Computation: Exploring the Power of Computing*. Boston, MA, USA, Addison-Wesley, 1998, 600 p.
9. *Processor Description Languages*. Ed. by M. Prabhat, D.Nikil. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2008, 432 p. https://doi.org/10.1016/B978-0-12-374287-2.X5001-0
10. Aarenstrup R. *Managing Model-Based Design*. Natick, MA, MathWorks Inc., 2015, 86 p.
11. Nane R., Sima V., Pilato C., Choi J., Fort B., Canis A., Chen Y., Hsiao H., Brown S., Ferrandi F., Anderson J., Bertels K. A Survey and evaluation of FPGA high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016, vol. 35, no. 10, pp. 1591–1604. https://doi.org/10.1109/TCAD.2015.2513673
12. Booch G., Jacobson I., Rumbaugh J. *The Unified Modeling Language User Guide*. Boston, MA, USA, Addison-Wesley, 1998, 391 p.
13. Delligatti L. *SysML Distilled: A Brief Guide to the Systems Modeling Language*. Boston, MA, USA, Addison-Wesley, 2013, 267 p.
14. Jean-Luc V. *Model-based System and Architecture Engineering with the Arcadia Method*. Elsevier, 2017, 388 p.

Автор

Горбачев Ярослав Георгиевич — инженер, ООО ЛМТ, Санкт-Петербург, 199034, Российская Федерация; преподаватель практики, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 57216337562, https://orcid.org/0000-0001-5419-6422, yaroslav-go@yandex.ru

Author

Yaroslav G. Gorbachev — Engineer, LMT Ltd., Saint Petersburg, 199034, Russian Federation; Practice Lecturer, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 57216337562, https://orcid.org/0000-0001-5419-6422, yaroslav-go@yandex.ru

Статья поступила в редакцию 15.05.2022
Одобрена после рецензирования 19.10.2022
Принята к печати 22.11.2022

Received 15.05.2022
Approved after reviewing 19.10.2022
Accepted 22.11.2022