Федеральное государственное бюджетное учреждение

Национальный исследовательский центр «Курчатовский институт»

Федеральное государственное учреждение «Федеральный научный центр

Научно-исследовательский институт системных исследований

Российской академии наук»

(ФГУ ФНЦ НИИСИ РАН)

ТРУДЫ НИИСИ РАН

ТОМ 13 № 4

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2023

Тематика номера:
Информационная безопасность, интернет вещей, полунатурное моделирование, архитектура АСУ ТП, вопросы программирования, проектирование и моделирование СБИС, высокопроизводительные вычисления на графических ускорителях, моделирование когнитивных процессов, информационные технологии в учебной информатике

Журнал публикует оригинальные статьи по следующим областям исследований: математика, математическое и компьютерное моделирование, обработка изображений, визуализация, системный анализ, методы обработки сигналов, информационная безопасность, информационные технологии, высокопроизводительные вычисления, оптико-нейронные технологии, микро- и наноэлектроника, математические исследования и вопросы численного анализа, история науки и техники.

The topic of the issue:

Information security, internet of things, HIL simulation, architecture of CPCS, programming issues, design and modeling of VLSI, high-performance computing on GPU, modeling of cognitive processes, information technology in educational informatics

The Journal publishes novel articles on the following research arias: mathematics, mathematical and computer modeling, image processing, visualization, system analysis, signal processing, information security, information technologies, high-performance computing, optical-neural technologies, micro- and nanoelectronics, mathematical researches and problems of numerical analysis, history of science and of technique.

Заведующий редакцией: В.Е.Текунов

Издатель: ФГУ ФНЦ НИИСИ РАН,
117218, Москва, Нахимовский проспект 36, к. 1

СОДЕРЖАНИЕ

CONTENT

Подход к определению типовых угроз безопасности информации для промышленных контроллеров

А. А. Асонов[1], А. И. Грюнталь[2], В. Н. Родионов[3]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, asonow@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, grntl@niisi.msk.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, rodionov@niisi.msk.ru

Аннотация. Определение актуальных угроз безопасности информации для некоторого объекта оценки, как правило, должно проводиться по Методике оценки угроз безопасности, утвержденной 5 февраля 2021 года в качестве методического документа ФСТЭК России. В настоящей статье показано, что при условии, когда к основному элементу объекта оценки, например, к ОС (в составе которого реализуются основные механизмы подсистемы защиты информации от несанкционированного доступа) и для которого существует введенный установленным порядком профиль защиты, перечень типовых угроз безопасности также можно получить из анализа данного документа. Сравнение угроз безопасности информации, полученных из профиля защиты операционных систем типа «В» четвертого класса защиты (ИТ.ОС.В4.ПЗ), с перечнем угроз безопасности информации, полученных из Базы данных ФСТЭК России на основе экспертного метода, после оптимизации данного перечня и исключения из него повторных угроз показывает, что оставшиеся из них находятся в определенном соответствии с угрозами из профиля защиты с некоторой детализацией по их реализации.

Ключевые слова: угроза безопасности информации (УБИ), пользователь УПК, АСУ ТП, операционная система

1. Введение

Под угрозой безопасности информации (УБИ) в соответствии с Методикой оценки угроз безопасности ФСТЭК России [2] понимается совокупность условий и факторов, создающих потенциальную или реально существующую опасность нарушения безопасности информации.

Определение типовых УБИ, актуальных для промышленного контроллера (ПК), является первым шагом по формированию (уточнению) требований безопасности и в первую очередь функциональных требований безопасности, реализуемых в его составе (на уровне ПК). Установленный перечень типовых УБИ может и должен быть также направлен на обоснование и выбор соответствующих организационных мер защиты информации.

В соответствии с требованиями ФСТЭК России как одного из регуляторов в области защиты информации, в технических заданиях (ТЗ) на разработку конкретных средств обработки информации или в целом автоматизированной системы должны задаваться требования безопасности информации.

При определении источников УБИ и выявлении потенциальных нарушителей и их возможностей по реализации УБИ проводится проверка их взаимосвязи с действующими методическими документами и указаниями ФСТЭК России по данному направлению работ.

В общем виде Методика оценки угроз безопасности [2] существует как методический документ ФСТЭК России, утвержденный 5 февраля 2021 года. Область применения данной методики распространяется на системы и сети, отнесенные к государственным и муниципальным информационным системам (ИС), ИС персональных данных, значимых объектов критической информационной инфраструктуры Российской Федерации, ИС управления производством, используемые организациями оборонно-промышленного комплекса (ОПК), автоматизированные системы управления производственными и технологическими процессами на критически важных объектах, потенциально опасных объектах и др. (методика введена в действие взамен ранее действующих соответствующих методик ФСТЭК России 2007 и 2008 годов). Особенностью данной методики является то, что в ней УБИ сформулированы и рассматриваются в целом (в общем виде) к системе (например, к АСУ ТП), а определить перечень УБИ типовых (актуальных), собственно, для некоторого элемента системы (в частности ПК) достаточно затруднительно.

В соответствии с Методикой [2] при определении перечня УБИ, типовых для УПК, необходимо также проанализировать перечень УБИ,

содержащийся в банке данных (БД) УБИ ФСТЭК России [6], из которого также должны быть выбраны только те угрозы, которые являются типовыми (актуальными), в частности, для ПК. Необходимо отметить, что в соответствии с Методикой [2] актуальность УБИ также должна определяться наличием и проверкой возможности по реализации сценариев их выполнения.

В целом выявленные актуальные УБИ для ПК могут и должны служить в качестве исходных данных, необходимых для разработки Модели УБИ системы и (или) сети (в частности, АСУ ТП) – требование Методики [2]. При необходимости они могут быть использованы для разработки Частной модели УБИ для ПК.

2. Анализ функциональных требований ПК как составной части АСУ ТП

2.1. Определение основных характеристик ПК, режимов работы и принципов информационного взаимодействия с другими составными частями системы (в частности элементами АСУ ТП)

Анализ базовой функциональной структуры системы с программируемым контроллером, модели аппаратного обеспечения программируемого контроллера, типовой конфигурации интерфейсов / портов ПК-системы, которые приведены в подразделе 4.1 (рис. 1 – 3) ГОСТ Р МЭК 61131-1-2016 [1], показывает следующее.

Аппаратное обеспечение ПК [1] включает: процессорные модули, модули системной шины и питания, модули ввода/вывода, модули передачи данных (необязательное оборудование), терминальные панели модулей ввода вывода, коммуникационные модули и кабели для подключения терминальных панелей к модулям ввода/вывода.

Программное обеспечение ПК [1], как правило, состоит из встраиваемого ПО в составе системного ПО (ОС контроллера, среда исполнения прикладного ПО и др.) и инструментального ПО (набор заголовочных файлов и библиотек ОС, компилятор, удаленный отладчик и др.).

На уровне инструментального ПО реализовываются различные режимы функционирования (режим конфигурирования и настройки ПК, режим наблюдения за работой системного и прикладного ПО, режим отладки, сервисный режим для обслуживания УПК, а также управление механизмами защиты информации (правами доступа пользователей), проверка корректности и целостности алгоритмов прикладного программного обеспечения (ППО), включая сравнение версий проектов и ряд других функций).

Центральной частью ПК является процессорный модуль, который как правило должен находиться под управлением многозадачной высокопроизводительной операционной системы реального времени (ОСРВ). ОСРВ с такими программами, как начальный загрузчик, входные/выходные данные, драйверы коммуникаций, обработчик исключительных ситуаций, планировщик, подсистема диагностики, подсистема управления резервированием и т. д., как правило, входит в состав неизменной части встраиваемого ПО ПК, а к изменяемой части встраиваемого программного обеспечения относится ППО, загружаемое в ПК пользователем.

То, что процессорный модуль (группа модулей) функционирует под управлением ОСРВ, позволяет считать, что основные требования безопасности информации, задаваемые к реализации в ПК, должны разрабатываться именно в составе ОСРВ, а отдельные требования БИ (в случае необходимости) – и на уровне других составных частей ПК.

Большую помощь в проведении данного анализа оказывает наличие технического задания на разработки конкретного ПК.

2.2. Определение потенциальных нарушителей функционирования ПК и их возможностей по реализации УБИ

В качестве непосредственного пользователя ПК, как правило, выступает ее авторизованный оператор или группа операторов, входящих в состав дежурной смены и должностных лиц по обеспечению функционирования, администрирования и обслуживания системы в целом, например, АСУ ТП или ее составной части (отдельного элемента). В целом возможности пользователей ПК (функции оператора и администратора безопасности) определяются возможностями их интерфейсов взаимодействия с ресурсами ОСРВ и средой ее функционирования.

В соответствии с Таблицей 8.1, определяющей уровни возможностей нарушителей по реализации УБИ, приведенной в Методике [3], представитель из данной группы специалистов с точки зрения оценки его возможностей по реализации УБИ потенциально может рассматриваться в качестве нарушителя, обладающего базовыми возможностями или базовыми повышенными возможностями (характеристики **Н1** и **Н2** соответственно). Потенциальные нарушители с характеристиками **Н3**, такие как разработчики программных и программно-аппаратных средств, не рассматриваются в связи с тем, что

после выполнения их работ созданные ими средства проходят все виды испытаний, включая сертификационные, а от выполнения дальнейших работ они как правило отключены. Данный тип нарушителя характеризуется как **внутренний** (относительно средств ПК).

Данные обстоятельства могут накладывать определенные условия и ограничения на выполняемые этими должностными лицами своих функциональных обязанностей по обеспечению функционирования ПК, а также состав УБИ, которые потенциально могут быть реализованы с их стороны. Необходимо также учитывать, что статус администратора безопасности информации ПК в ряде случаев может выводить соответствующее должностное лицо из числа потенциальных нарушителей (может указываться в техническом задании на разработку конкретного ПК, а также определяться из анализа требований безопасности информации по разработке конкретной автоматизированной системы управления технологическими процессами или ее подсистемы).

Установление и согласование с Заказчиком ОКР возможностей нарушителей и их характеристик позволяет разработчику осуществить уточнение состава УБИ, по отношению к которым необходимо проводить мероприятия по защите информации.

2.3 Определение требований безопасности информации, предъявляемых к ПК со стороны системы (например – АСУ ТП)

Анализ требований безопасности информации, предъявляемых в частности к АСУ ТП (для наиболее высокого класса защищенности), в которых применяется ПК, показывает, что такая система должна обеспечивать защищенность до **1 класса защиты** включительно согласно приказу ФСТЭК России от 14 марта 2014 г. № 31 [7], а также в составе значимых объектов критической информационной инфраструктуры до **1 категории значимости** включительно согласно приказу ФСТЭК России от 25 декабря 2017 г. № 239 [8].

Функциональное назначение, принцип обработки информации и место ПК в системе (АСУ ТП) показывает, что уровень конфиденциальности обрабатываемой информации, как правило, характеризует ее как информацию ограниченного доступа («для служебного пользования»), что соответствует требованиям «4 уровню доверия» согласно Приказу ФСТЭК России от 02 июня 2020 г. № 76 [9].

В подразделе 2.1 настоящей статьи сделан вывод, что основным элементом ПК является операционная система (ОСРВ), на уровне которой должны реализовываться основные функции и компоненты безопасности.

Из вышеуказанных требований, вытекает то, что для такого класса защиты АСУ ТП (**1 класс защищенности и 4 уровень доверия**) в соответствии с п. 8 и п. 9 абзаца 4 Приказа ФСТЭК России от 19 августа 2016 г. № 119 [4] им наиболее полно соответствует **ОС 4 класса защиты тип «В»** (ОСРВ).

Рассмотрение и сравнение состава функций безопасности (ФБ), реализуемых с составе ОС типа «В» (ОСРВ) и ОС типа «Б» (встраиваемые ОС), указанных в Таблице 1 Приказа ФСТЭК России от 19 августа 2016 г. № 119, показывает следующее:

- в ОС типа «В» должны быть реализованы все ФБ1 – ФБ8 (ФБ1 – идентификация и аутентификация, ФБ2 – управление доступом, ФБ3 – регистрация событий безопасности, ФБ4 – ограничение программной среды, ФБ5 – изоляция процессов, ФБ6 – защита памяти, ФБ7 – контроль целостности компонентов ОС, а также иных объектов файловой системы, ФБ8 – обеспечение надежного функционирования);
- в ОС типа «Б» из ФБ1 – ФБ8 отсутствуют требования по реализации ФБ5 и ФБ7.

Частный вывод: ОС типа «В» по сравнению с ОС типа «Б» в части предъявляемых и реализуемых ФБ является более функционально полной.

3. Определение перечня типовых (актуальных) угроз безопасности информации ПК

Однозначное представление о типовых (актуальных) УБИ, которым должна противостоять ОСРВ, дает профиль защиты [5], в нашем случае Методический документ ФСТЭК России «Профиль защиты (ПЗ) операционных систем типа «В» четвертого класса защиты» – ИТ.ОС.В4.ПЗ [4]. Среди угроз, которым должна противостоять ОСРВ, угрозы У1 – У11 и угрозы, которым должна противостоять среда функционирования ОСРВ, УС1 – УС7 (обозначение угроз дано в соответствии с ПЗ).

С другой стороны, возникает необходимость проверить, как угрозы из профиля защиты соотносятся с угрозами БИ, приведенными в базе данных угроз (БД) ФСТЭК России.

Все УБИ, представленные в БД угроз (222 угрозы), были рассмотрены и из них были выбраны только те, которые могут оказывать влияние на функционирование ПК (отбор проводился в соответствии с характеристиками, указанными в БД: **«объект воздействия»** – это ПК (один из элементов АСУ ТП); **«источник угрозы»** – это внутренний нарушитель – ВН с

низким потенциалом – **Н1** и средним потенциалом – **Н2** (из-за принятого в подразделе 2.2 соглашения внутренний нарушитель с высоким потенциалом – **Н3** не рассматривается). Проведение выбора из приведенных угроз в БД осуществлялось на основе экспертного метода специалистами разработчика. При необходимости дополнительная минимизация УБИ может проводиться по характеристике БД **«последствия реализации угрозы»** (нарушение конфиденциальности, целостности, доступности – выбору подлежат только те УБИ из БД, для которых эта характеристика задана или является существенной).

Все данные по выбору УБИ из ПЗ и их сопоставлению с УБИ из БД ФСТЭК России отображены в Таблице 1 (см. Приложение к статье).

В Таблице 1 представлены УБИ, выбранные из ПЗ, в сопоставлении с УБИ, выбранными из Базы данных УБИ ФСТЭК России [6] (второй столбец таблицы). В третьем столбце Таблицы 1 за каждой УБИ из ПЗ отражены УБИ из базы как наиболее типовые (актуальные) для ПК. Порядок выбора был следующим: из общего числа выбранных угроз из базы, если они повторялись по отношению к У1 – У11 и УС1 – УС7, в третьем столбце таблицы отражается только одна угроза (например, УБИ.015 (отмечена экспертом) соответствует У1 и УС2, но в последующем для У1 она была убрана (исключена из рассмотрения в связи с повтором), а для УС2 оставлена как наиболее характерная для этого случая). Данный подход выполнялся для всех УБИ из базы.

Предполагается, что выполнение данного анализа позволит более детально рассмотреть угрозы БИ из БД с точки зрения их устранения при реализации конкретной архитектуры ПК (ее составных частей) с учетом условий эксплуатации создаваемого изделия, отмеченных в эксплуатационной документации в рамках установленных испытаний образца и в ходе его сертификационных испытаний (например, устранение «УБИ.165: Угроза включения в проект не достоверно испытанных компонентов», как правило, проверяется в рамках приемочных испытаний опытного образца изделия).

При этом необходимо учитывать, что отдельные угрозы парируются организационными или организационно-техническими мерами (например, У5, (УБИ012) – конкретной регламентацией (до отдельного параметра) за конфигурированием ПО (ОС) и организацией контроля за выполнением данных работ и др.).

При определении типовых (актуальных) угроз БИ, необходимо учитывать такой факт, что в ПК, как правило, одновременно могут существовать не более двух пользователей (см. замечание выше), функциональные обязанности которых предполагают все аспекты деятельности практически со всеми его ресурсами.

Анализ угроз из БД ФСТЭК России показывает, что определенная их часть также должна дополнительно рассматриваться и решаться интегратором при создании конкретного АСУ ТП (при встраивании ПК в систему в качестве ее составной части), в частности при разработке частных руководств и инструкций по его применению.

При проведении данной работы в практическом плане необходимо учитывать и то, что часть УБИ сформированы таким образом, что они должны «парироваться» исключительно организационными мерами.

Выполнение минимизации перечня УБИ позволит сократить его не менее чем в два-три раза и упростит их дальнейшее рассмотрение.

4. Заключение

Приведенное в Таблице 1 сравнение угроз безопасности информации, полученных из ПЗ, с перечнем угроз безопасности информации, полученных из БД ФСТЭК России на основе экспертного метода, после оптимизации данного перечня и исключения из него повторных угроз показывает, что оставшиеся из них находятся в очевидном соответствии с угрозами из профиля защиты с некоторой детализацией по их реализации.

В целом предложенный подход по определению УБИ, типовых для конкретного элемента автоматизированной системы (в частности ПК), базирующийся на наличии и использовании исходных данных заказчика, а также наличие НМД (нормативно-методической документации) Регулятора в области защиты (в статье это профиль защиты для ОС типа «В» 4 класса защиты и УБИ БД ФСТЭК России) позволяет значительно сократить время для выполнения этой работы.

С точки зрения эффективности данный подход позволяет получить результаты аналогичные, но не хуже по сравнению с методом, приведенным в Методическом документе ФСТЭК России «Методика оценки угроз безопасности информации», утвержденном ФСТЭК России 5 февраля 2021 года (подход и метод базируются на проведении экспертных оценок).

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).

ПРИЛОЖЕНИЕ

Таблица 1. Соответствие УБИ из ПЗ угрозам безопасности из Базы данных УБИ ФСТЭК России

УБИ из ПЗ (указатель и аннотация), характеристика нарушителя (ВН – внутренний, ВНЕШ – внешний)	**УБИ из БД ФСТЭК России (указатель и аннотация)**	**УБИ из БД ФСТЭК России (указатель и аннотация) с учетом исключения повторных УБИ**
У1 – НСД к объектам доступа со стороны субъектов доступа, для которых запрашиваемый доступ не разрешен (ВН)	УБИ.015: Угроза доступа к защищаемым файлам с использованием обходного пути УБИ.028: Угроза использования альтернативных путей доступа к ресурсам УБИ.033: Угроза использования слабостей кодирования входных данных УБИ.034: Угроза использования слабостей протоколов сетевого/локального обмена данными УБИ.036: Угроза исследования механизмов работы программы УБИ.037: Угроза исследования приложения через отчеты об ошибках УБИ.067: Угроза неправомерного ознакомления с защищаемой информацией УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.073: Угроза несанкционированного доступа к активному и (или) пассивному виртуальному и (или) физическому сетевому оборудованию из физической и (или) виртуальной сети УБИ.074: Угроза несанкционированного доступа к аутентификационной информации УБИ.075: Угроза несанкционированного доступа к виртуальным каналам передачи УБИ.076: Угроза несанкционированного доступа к гипервизору из виртуальной машины и (или) физической сети УБИ.077: Угроза несанкционированного доступа к данным за пределами зарезервированного адресного пространства, в том числе выделенного под виртуальное аппаратное обеспечение УБИ.080: Угроза несанкционированного доступа к защищаемым	УБИ.033: Угроза использования слабостей кодирования входных данных УБИ.067: Угроза неправомерного ознакомления с защищаемой информацией УБИ.073: Угроза несанкционированного доступа к активному и (или) пассивному виртуальному и (или) физическому сетевому оборудованию из физической и (или) виртуальной сети УБИ.075: Угроза несанкционированного доступа к виртуальным каналам передачи УБИ.076: Угроза несанкционированного доступа к гипервизору из виртуальной машины и (или) физической сети УБИ.077: Угроза несанкционированного доступа к данным за пределами зарезервированного адресного пространства, в том числе выделенного под виртуальное аппаратное обеспечение УБИ.080: Угроза несанкционированного доступа к защищаемым виртуальным устройствам из виртуальной и (или) физической сети УБИ.083: Угроза несанкционированного доступа к системе по беспроводным каналам УБИ.084: Угроза несанкционированного доступа к системе хранения данных из виртуальной и (или) физической сети УБИ.085: Угроза несанкционированного доступа к хранимой в виртуальном пространстве защищаемой информации УБИ.090: Угроза несанкционированного создания учетной записи пользователя УБИ.092: Угроза несанкционированного удаленного внеполосного доступа к аппаратным средствам УБИ.207: Угроза несанкционированного доступа к параметрам

	виртуальным устройствам из виртуальной и (или) физической сети УБИ.083: Угроза несанкционированного доступа к системе по беспроводным каналам УБИ.084: Угроза несанкционированного доступа к системе хранения данных из виртуальной и (или) физической сети УБИ.085: Угроза несанкционированного доступа к хранимой в виртуальном пространстве защищаемой информации УБИ.090: Угроза несанкционированного создания учетной записи пользователя УБИ.092: Угроза несанкционированного удаленного внеполосного доступа к аппаратным средствам УБИ.116: Угроза перехвата данных, передаваемых по вычислительной сети УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя УБИ.131: Угроза подмены субъекта сетевого доступа УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.207: Угроза несанкционированного доступа к параметрам настройки оборудования за счет использования «мастер-кодов» (инженерных паролей) УБИ.209: Угроза несанкционированного доступа к защищаемой памяти ядра процессора УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	настройки оборудования за счет использования «мастер-кодов» (инженерных паролей) УБИ.209: Угроза несанкционированного доступа к защищаемой памяти ядра процессора

	УБИ.215: Угроза несанкционированного доступа к системе при помощи сторонних сервисов	
У2 – ограничение нарушителем доступа пользователей ОС к ресурсам СВТ, на котором установлена ОС за счет длительного удержания вычислительного ресурса в загруженном состоянии путем осуществления нарушителем многократных запросов, требующих большого количества ресурсов на их обработку (ВН)	УБИ.014: угроза длительного удержания вычислительных ресурсов пользователями (ВН, ВНЕШ). Объекты доступа – сетевой трафик, сетевое ПО, сетевой узел, системное ПО УБИ.028: Угроза использования альтернативных путей доступа к ресурсам УБИ.034: Угроза использования слабостей протоколов сетевого/локального обмена данными УБИ.059: Угроза неконтролируемого роста числа зарезервированных вычислительных ресурсов УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.069: Угроза неправомерных действий в каналах связи УБИ.098: Угроза обнаружения открытых портов и идентификации привязанных к ним сетевых служб УБИ.128: Угроза подмены доверенного пользователя УБИ.140: Угроза приведения системы в состояние «отказ в обслуживании» УБИ.153: Угроза усиления воздействия на вычислительные ресурсы пользователей при помощи сторонних серверов УБИ.155: Угроза утраты вычислительных ресурсов УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.169: Угроза наличия механизмов разработчика УБИ.176: Угроза нарушения технологического/производственного процесса из-за временны́х задержек, вносимых средством защиты УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уяз-	УБИ.028: Угроза использования альтернативных путей доступа к ресурсам УБИ.034: Угроза использования слабостей протоколов сетевого/локального обмена данными УБИ.098: Угроза обнаружения открытых портов и идентификации привязанных к ним сетевых служб УБИ.153: Угроза усиления воздействия на вычислительные ресурсы пользователей при помощи сторонних серверов

	вимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.208: Угроза нецелевого использования вычислительных ресурсов средства вычислительной техники УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
У3 – несанкционированное или ошибочное удаление информации с СВТ, функционирующего под управлением ОС (ВН)	УБИ.034: Угроза использования слабостей протоколов сетевого/локального обмена данными УБИ.036: Угроза исследования механизмов работы программы УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.069: Угроза неправомерных действий в каналах связи УБИ.091: Угроза несанкционированного удаления защищаемой информации УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя УБИ.143: Угроза программного выведения из строя средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.152: Угроза удаления аутентификационной информации УБИ.156: Угроза утраты носителей информации УБИ.157: Угроза физического выведения из строя средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.158: Угроза форматирования носителей информации УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.169: Угроза наличия механизмов разработчика УБИ.182: Угроза физического устаревания аппаратных компонентов	УБИ.091: Угроза несанкционированного удаления защищаемой информации УБИ.143: Угроза программного выведения из строя средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.152: Угроза удаления аутентификационной информации УБИ.158: Угроза форматирования носителей информации

	УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
У4 – утечка или несанкционированное изменение информации в ОП, используемой различными процессами и формируемыми ими потоками (ВН и ВНЕШ)	УБИ.025: Угроза изменения системных и глобальных переменных УБИ.026: Угроза искажения XML-схемы УБИ.036: Угроза исследования механизмов работы программы УБИ.037: Угроза исследования приложения через отчеты об ошибках УБИ.069: Угроза неправомерных действий в каналах связи УБИ.086: Угроза несанкционированного изменения аутентификационной информации УБИ.116: Угроза перехвата данных, передаваемых по вычислительной сети УБИ.128: Угроза подмены доверенного пользователя УБИ.132: Угроза получения предварительной информации об объекте защиты УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.169: Угроза наличия механизмов разработчика УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.185: Угроза несанкционированного изменения параметров	УБИ.025: Угроза изменения системных и глобальных переменных УБИ.132: Угроза получения предварительной информации об объекте защиты УБИ.169: Угроза наличия механизмов разработчика УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.193: Угроза утечки информации за счет применения вредоносным программным обеспечением алгоритмов шифрования трафика УБИ.203: Угроза утечки информации с неподключенных к сети Интернет компьютеров

	настройки средств защиты информации УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.193: Угроза утечки информации за счет применения вредоносным программным обеспечением алгоритмов шифрования трафика УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.203: Угроза утечки информации с неподключенных к сети Интернет компьютеров УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
У5 – несанкционированное внесение нарушителем изменений в конфигурационные (и иные) данные, которые влияют на функционирование отдельных сервисов, приложений или ОС в целом (ВН)	УБИ.012 – угроза деструктивного изменения конфигурации/среды окружения ПО Объекты доступа – системное ПО, ППО, сетевое ПО, микропрограммное обеспечение УБИ.023: Угроза изменения компонентов информационной (автоматизированной) системы УБИ.025: Угроза изменения системных и глобальных переменных УБИ.026: Угроза искажения XML-схемы УБИ.036: Угроза исследования механизмов работы программы УБИ.037: Угроза исследования приложения через отчеты об ошибках УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.169: Угроза наличия механизмов разработчика	УБИ.012 – угроза деструктивного изменения конфигурации/среды окружения ПО Объекты доступа – системное ПО, ППО, сетевое ПО, микропрограммное обеспечение УБИ.023: Угроза изменения компонентов информационной (автоматизированной) системы УБИ.026: Угроза искажения XML-схемы УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.185: Угроза несанкционированного изменения параметров настройки средств защиты информации УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.204: Угроза несанкционированного изменения вредоносной программой значений параметров программируемых логических контроллеров

	УБИ.179: Угроза несанкционированной модификации защищаемой информации УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.185: Угроза несанкционированного изменения параметров настройки средств защиты информации УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.204: Угроза несанкционированного изменения вредоносной программой значений параметров программируемых логических контроллеров УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
У6 – осуществление восстановления (подбора) аутентификационной информации пользователей ОС (ВН и ВНЕШ)	УБИ.008: Угроза восстановления и/или повторного использования аутентификационной информации УБИ.030: Угроза использования информации идентификации/аутентификации, заданной по умолчанию УБИ.034: Угроза использования слабостей протоколов сетевого/локального обмена данными УБИ.036: Угроза исследования механизмов работы программы УБИ.067: Угроза неправомерного ознакомления с защищаемой информацией УБИ.074: Угроза несанкционированного доступа к аутентификационной информации УБИ.100: Угроза обхода некорректно настроенных механизмов аутентификации УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации	УБИ.008: Угроза восстановления и/или повторного использования аутентификационной информации УБИ.213: Угроза обхода многофакторной аутентификации

	УБИ.169: Угроза наличия механизмов разработчика УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.213: Угроза обхода многофакторной аутентификации УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
У7 – использование нарушителем идентификационной и начальной аутентификационной информации, соответствующей учетной записи пользователя ОС (ВН и ВНЕШ)	УБИ.030 – угроза использования информации идентификации/аутентификации заданной по умолчанию УБИ.067: Угроза неправомерного ознакомления с защищаемой информацией УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.169: Угроза наличия механизмов разработчика УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	УБИ.030 – угроза использования информации идентификации/аутентификации заданной по умолчанию УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя
У8 – несанкционированное внесение изменений в журналы регистрации событий безопасности ОС (ВН)	УБИ.037: Угроза исследования приложения через отчеты об ошибках УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.124: Угроза подделки записей журнала регистрации событий УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя УБИ.160: Угроза хищения средств	УБИ.037: Угроза исследования приложения через отчеты об ошибках УБИ.179: Угроза несанкционированной модификации защищаемой информации

	хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.169: Угроза наличия механизмов разработчика УБИ.179: Угроза несанкционированной модификации защищаемой информации УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
У9 – НСД к информации вследствие использования пользователями ОС неразрешенного ПО (ВН)	УБИ.067: Угроза неправомерного ознакомления с защищаемой информацией УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.116: Угроза перехвата данных, передаваемых по вычислительной сети УБИ.132: Угроза получения предварительной информации об объекте защиты УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкционированного воздействия на средство защиты информации	УБИ.188: Угроза подмены программного обеспечения УБИ.191: Угроза внедрения вредоносного кода в дистрибутив программного обеспечения УБИ.211: Угроза использования непроверенных пользовательских данных при формировании конфигурационного файла, используемого программным обеспечением администрирования информационных систем УБИ.217: Угроза использования скомпрометированного доверенного источника обновлений программного обеспечения

	УБИ.188: Угроза подмены программного обеспечения УБИ.189: Угроза маскирования действий вредоносного кода УБИ.191: Угроза внедрения вредоносного кода в дистрибутив программного обеспечения УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.211: Угроза использования непроверенных пользовательских данных при формировании конфигурационного файла, используемого программным обеспечением администрирования информационных систем УБИ.217: Угроза использования скомпрометированного доверенного источника обновлений программного обеспечения УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
У10 – НСБ субъектов доступа к информации, обработка которой осуществлялась в рамках сеансов (сессий) других субъектов доступа (ВН)	УБИ.034: Угроза использования слабостей протоколов сетевого/локального обмена данными УБИ.067: Угроза неправомерного ознакомления с защищаемой информацией УБИ.074: Угроза несанкционированного доступа к аутентификационной информации УБИ.116: Угроза перехвата данных, передаваемых по вычислительной сети УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя УБИ.131: Угроза подмены субъекта сетевого доступа УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.192: Угроза использования уязвимых версий программного обеспечения	УБИ.116: Угроза перехвата данных, передаваемых по вычислительной сети УБИ.131: Угроза подмены субъекта сетевого доступа УБИ.215: Угроза несанкционированного доступа к системе при помощи сторонних сервисов

	УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.215: Угроза несанкционированного доступа к системе при помощи сторонних сервисов	
У11 – недоступность вычислительных ресурсов (процессорное время, ОП и др.) для критических служб ОС и функционирующего ППО (приложений) вследствие нерационального распределения ресурсов между потоками служб и приложений (без учета степени критичности) (ВН)	УБИ.014: Угроза длительного удержания вычислительных ресурсов пользователями УБИ.022 – угроза избыточного выделения ОП УБИ.038: Угроза исчерпания вычислительных ресурсов хранилища больших данных УБИ.059: Угроза неконтролируемого роста числа зарезервированных вычислительных ресурсов УБИ.069: Угроза неправомерных действий в каналах связи УБИ.098: Угроза обнаружения открытых портов и идентификации привязанных к ним сетевых служб УБИ.128: Угроза подмены доверенного пользователя УБИ.140: Угроза приведения системы в состояние «отказ в обслуживании» УБИ.155: Угроза утраты вычислительных ресурсов УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.166: Угроза внедрения системной избыточности УБИ.169: Угроза наличия механизмов разработчика УБИ.176: Угроза нарушения технологического/производственного процесса из-за временны́х задержек, вносимых средством защиты УБИ.179: Угроза несанкционированной модификации защищаемой информации УБИ.182: Угроза физического устаревания аппаратных компонентов УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода	УБИ.014: Угроза длительного удержания вычислительных ресурсов пользователями УБИ.022 – угроза избыточного выделения ОП УБИ.038: Угроза исчерпания вычислительных ресурсов хранилища больших данных УБИ.059: Угроза неконтролируемого роста числа зарезервированных вычислительных ресурсов УБИ.140: Угроза приведения системы в состояние «отказ в обслуживании» УБИ.155: Угроза утраты вычислительных ресурсов УБИ.208: Угроза нецелевого использования вычислительных ресурсов средства вычислительной техники

	УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.208: Угроза нецелевого использования вычислительных ресурсов средства вычислительной техники УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
УС1 – нарушение целостности программных компонентов ОС (ВН и ВНЕШ)	УБИ.026: Угроза искажения XML-схемы УБИ.036: Угроза исследования механизмов работы программы УБИ.037: Угроза исследования приложения через отчеты об ошибках УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.069: Угроза неправомерных действий в каналах связи УБИ.143: Угроза программного выведения из строя средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.156: Угроза утраты носителей информации УБИ.157: Угроза физического выведения из строя средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.169: Угроза наличия механизмов разработчика УБИ.179: Угроза несанкционированной модификации защищаемой информации УБИ.182: Угроза физического устаревания аппаратных компонентов УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкциониро-	УБИ.036: Угроза исследования механизмов работы программы УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.069: Угроза неправомерных действий в каналах связи УБИ.156: Угроза утраты носителей информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов

	ванного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
УС2 – отключение и (или) обход нарушителями компонентов ОС, реализующих функции БИ путем подмены нарушителями загружаемой ОС (ВН и ВНЕШ)	УБИ.015 – угроза доступа к защищенным файлам с использованием обходного пути УБИ.018 – угроза связанная с подменой ОС при загрузке путем несанкционированного переконфигурирования BIOS|UEF1 пути доступа к загрузчику ОС УБИ.034: Угроза использования слабостей протоколов сетевого/локального обмена данными УБИ.036: Угроза исследования механизмов работы программы УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.179: Угроза несанкционированной модификации защищаемой информации УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного за-	УБИ.015 – угроза доступа к защищенным файлам с использованием обходного пути УБИ.018 – угроза связанная с подменой ОС при загрузке путем несанкционированного переконфигурирования BIOS|UEF1 пути доступа к загрузчику ОС УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации

	пуска вредоносного кода в обход механизмов защиты операционной системы УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
УС3 – нарушение целостности данных (в т. ч. параметров настройки СЗИ ОС (ВН и ВНЕШ)	УБИ.049: Угроза нарушения целостности данных кеша УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.143: Угроза программного выведения из строя средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.145: Угроза пропуска проверки целостности программного обеспечения УБИ.156: Угроза утраты носителей информации УБИ.157: Угроза физического выведения из строя средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.169: Угроза наличия механизмов разработчика УБИ.179: Угроза несанкционированной модификации защищаемой информации УБИ.182: Угроза физического устаревания аппаратных компонентов УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.214: Угроза несвоевременного	УБИ.049: Угроза нарушения целостности данных кеша УБИ.145: Угроза пропуска проверки целостности программного обеспечения УБИ.157: Угроза физического выведения из строя средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.187: Угроза несанкционированного воздействия на средство защиты информации

	выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
УС4 – НСД нарушителя к аутентификационной информации администраторов и (или) пользователей ОС (ВН и ВНЕШ)	УБИ.008: Угроза восстановления и/или повторного использования аутентификационной информации УБИ.030: Угроза использования информации идентификации/аутентификации, заданной по умолчанию УБИ.036: Угроза исследования механизмов работы программы УБИ.067: Угроза неправомерного ознакомления с защищаемой информацией УБИ.074: Угроза несанкционированного доступа к аутентификационной информации УБИ.086: Угроза несанкционированного изменения аутентификационной информации УБИ.090: Угроза несанкционированного создания учетной записи пользователя УБИ.100: Угроза обхода некорректно настроенных механизмов аутентификации УБИ.116: Угроза перехвата данных, передаваемых по вычислительной сети УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.198: Угроза скрытной регистрации вредоносной программой учетных записей администраторов УБИ.214: Угроза несвоевременного	УБИ.074: Угроза несанкционированного доступа к аутентификационной информации УБИ.086: Угроза несанкционированного изменения аутентификационной информации УБИ.100: Угроза обхода некорректно настроенных механизмов аутентификации УБИ.198: Угроза скрытной регистрации вредоносной программой учетных записей администраторов

	выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
УС5 – несанкционированное внесение нарушителем изменений в журналы регистрации событий безопасности ОС за счет доступа к файлам журналов регистрации событий безопасности ОС в среде функционирования ОС с использованием специальных программных средств, предоставляющих возможность обрабатывать файлы журналов регистрации событий безопасности ОС (ВН)	УБИ.037: Угроза исследования приложения через отчеты об ошибках УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.124: Угроза подделки записей журнала регистрации событий УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя УБИ.169: Угроза наличия механизмов разработчика УБИ.179: Угроза несанкционированной модификации защищаемой информации УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	УБИ.124: Угроза подделки записей журнала регистрации событий
УС6 – несанкционированное копирование информации из памяти СВТ на съемные МНИ (или другое место вне информационной системы) пользователем ОС (ВН)	УБИ.034: Угроза использования слабостей протоколов сетевого/локального обмена данными УБИ.057: Угроза неконтролируемого копирования данных внутри хранилища больших данных УБИ.067: Угроза неправомерного ознакомления с защищаемой информацией УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.069: Угроза неправомерных	УБИ.057: Угроза неконтролируемого копирования данных внутри хранилища больших данных УБИ.088: Угроза несанкционированного копирования защищаемой информации УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации

	действий в каналах связи УБИ.088: Угроза несанкционированного копирования защищаемой информации УБИ.116: Угроза перехвата данных, передаваемых по вычислительной сети УБИ.127: Угроза подмены действия пользователя путем обмана УБИ.128: Угроза подмены доверенного пользователя УБИ.132: Угроза получения предварительной информации об объекте защиты УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.169: Угроза наличия механизмов разработчика УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	
УС7 – снижение производительности ОС из-за внедрения в нее избыточного ПО и его компонентов (ВН)	УБИ.022: Угроза избыточного выделения оперативной памяти УБИ.036: Угроза исследования механизмов работы программы УБИ.059: Угроза неконтролируемого роста числа зарезервированных вычислительных ресурсов УБИ.068: Угроза неправомерного/некорректного использования интерфейса взаимодействия с приложением УБИ.127: Угроза подмены действия пользователя путем обмана	УБИ.161: Угроза чрезмерного использования вычислительных ресурсов суперкомпьютера в ходе интенсивного обмена межпроцессорными сообщениями УБИ.166: Угроза внедрения системной избыточности УБИ.176: Угроза нарушения технологического/производственного процесса из-за временны́х задержек, вносимых средством защиты УБИ.180: Угроза отказа подсистемы обеспечения температурного режима

	УБИ.155: Угроза утраты вычислительных ресурсов УБИ.160: Угроза хищения средств хранения, обработки и (или) ввода/вывода/передачи информации УБИ.161: Угроза чрезмерного использования вычислительных ресурсов суперкомпьютера в ходе интенсивного обмена межпроцессорными сообщениями УБИ.165: Угроза включения в проект не достоверно испытанных компонентов УБИ.166: Угроза внедрения системной избыточности УБИ.169: Угроза наличия механизмов разработчика УБИ.176: Угроза нарушения технологического/производственного процесса из-за временны́х задержек, вносимых средством защиты УБИ.179: Угроза несанкционированной модификации защищаемой информации УБИ.180: Угроза отказа подсистемы обеспечения температурного режима УБИ.182: Угроза физического устаревания аппаратных компонентов УБИ.183: Угроза перехвата управления автоматизированной системой управления технологическими процессами УБИ.187: Угроза несанкционированного воздействия на средство защиты информации УБИ.189: Угроза маскирования действий вредоносного кода УБИ.192: Угроза использования уязвимых версий программного обеспечения УБИ.195: Угроза удаленного запуска вредоносного кода в обход механизмов защиты операционной системы УБИ.208: Угроза нецелевого использования вычислительных ресурсов средства вычислительной техники УБИ.214: Угроза несвоевременного выявления и реагирования компонентами информационной (автоматизированной) системы (в том числе средствами защиты информации) на события безопасности информации	УБИ.182: Угроза физического устаревания аппаратных компонентов

An Approach to Identifying Information Security Threats Relevant and Specific for a Universal Industrial Controller

Alexander Asonov, Andrey Gruntal, Victor Rodionov

Abstract. Determining current threats to information security for a certain object of assessment, as a rule, should be carried out according to the Methodology for Assessing Security Threats, approved on February 5, 2021 as a methodological document of the FSTEC of Russia. This article shows that, provided that the main element of the assessment object, e.g., OS, (which implements the main mechanisms of the subsystem for protecting information from unauthorized access) and for which there is a security profile introduced in the established order, the list of typical security threats can also be obtained from the analysis of this document. Comparison of information security threats obtained from the security profile of operating systems of type "B" of the 4th security class, with a list of information security threats obtained from the FSTEC of Russia Database based on the expert method, after optimizing this list and the exclusion of repeated threats from it, shows that the remaining threats are in certain correspondence with the threats from the security profile with some detail on their implementation.

Keywords: information security threat, universal industrial controller, automated process control system, operating system

Литература

1. Контроллеры программируемые. Часть 1. Общая информация (IEC 61131-1:2003, IDT). URL: https://docs.cntd.ru/document/1200135007 (дата обращения 13.10.2023).

2. «Методика оценки угроз безопасности информации». Методический документ ФСТЭК России, утвержден 5 февраля 2021 года. URL: https://fstec.ru/dokumenty/vse-dokumenty/spetsialnye-normativnye-dokumenty/metodicheskij-dokument-ot-5-fevralya-2021-g (дата обращения 13.10.2023).

3. Приказ ФСТЭК России от 9 августа 2018 г. № 138 «О внесении изменений в требования к обеспечению защиты информации в автоматизированных системах управления производственными и технологическими процессами на критически важных объектах, потенциально опасных объектах, а также объектах, представляющих повышенную опасность для жизни и здоровья людей и окружающей природной среды, утвержденные приказом ФСТЭК России от 14 марта 2014 г. № 31, и в требования по обеспечению безопасности значимых объектов критической информационной инфраструктуры Российской Федерации, утвержденными приказом ФСТЭК России от 25 декабря 2017 г. № 239».

4. Приказ ФСТЭК России от 19 августа 2016 г. № 119 «Требования в области технического регулирования к продукции, используемой в целях защиты сведений, составляющих государственную тайну или относимых к охраняемой в соответствии с законодательством Российской Федерации ограниченного доступа (требования безопасности информации к операционным системам)».

5. Методический документ. «Профиль защиты операционных систем типа «В» четвертого класса защиты – ИТ.ОС.В4.ПЗ». https://www.garant.ru/products/ipo/prime/doc/71588736/ (дата обращения 13.10.2023).

6. «Банк данных угроз безопасности информации ФСТЭК России». URL: https://bdu.fstec.ru (дата обращения 13.10.2023).

7. Приказ ФСТЭК России от 14 марта 2014 г. № 31 «Об утверждении требований к обеспечению защиты информации в автоматизированных системах управления производственными и технологическими процессами на критически важных объектах, потенциально опасных объектах, а также объектах, представляющих повышенную опасность для жизни и здоровья людей и для окружающей природной среды». URL: https://fstec.ru/dokumenty/vse-dokumenty/prikazy/prikaz-fstek-rossii-ot-14-marta-2014-g-n-31 (дата обращения 13.10.2023).

8. Приказ ФСТЭК России от 25 декабря 2017 г. № 239 «Об утверждении требований по обеспечению безопасности значимых объектов критической информационной инфраструктуры российской федерации». URL: https://fstec.ru/dokumenty/vse-dokumenty/prikazy/prikaz-fstek-rossii-ot-25-dekabrya-2017-g-n-239 (дата обращения 13.10.2023).

9. Приказ ФСТЭК России № 76 от 02.06.2020 «Об утверждении Требований по безопасности информации, устанавливающих уровни доверия к технической защиты информации и средствам

обеспечения безопасности информационных технологий». URL: https://check-ib.ru/docs/prikaz-fstek-rossii-76-ot-02-06-2020/ (дата обращения 13.10.2023).

Использование аппаратных средств профилирования для обеспечения информационной безопасности критически важных систем

В. А. Галатенко[1], К. А. Костюхин[2]

[1]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, galat@niisi.ras.ru;

[2]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, kost@niisi.ras.ru

Аннотация. Работа посвящена исследованию возможностей применения аппаратных счетчиков производительности (специальных регистров центрального процессора) для выявления потенциальных угроз безопасности критически важных систем и комплексов. Авторами был доработан открытый прикладной программный интерфейс измерения производительности, с помощью которого осуществляется управление аппаратными счетчиками.

Ключевые слова: счетчики производительности, информационная безопасность, атаки по сторонним каналам, PAPI

1. Введение

Многие современные процессоры поддерживают методы анализа программного кода за счет использования аппаратных счетчиков (специальных регистров, записывающих определенные типы аппаратных событий). К примерам аппаратных событий относятся общее количество циклов процессора, общее количество выполненных инструкций, количество выполненных операций с плавающей запятой, количество промахов при доступе к кэш-памяти и т. д. Изначально аппаратные счетчики использовались специально для построения профилей выполнения и последующей оптимизации, но они могут выполнять и другую важную функцию — помогать разработчикам и системным архитекторам оперативно выявлять так называемые атаки по сторонним каналам (side-channels attacks [1]). В рамках этой работы была исследована возможность использования аппаратных счетчиков для обнаружения таких атак и адаптирован программный интерфейс измерения производительности (Performance Application Programming Interface, PAPI [2]) для аппаратной платформы, работающей под управлением отечественного операционной системы.

2. Архитектура PAPI

Целью проекта PAPI является разработка, стандартизация и внедрение портативного и эффективного интерфейса прикладного программирования для доступа к аппаратным средствам профилирования. Сегодня PAPI стал стандартом де-факто для разработчиков программного обеспечения, имеющего доступ к аппаратным счетчикам производительности.

На рисунке 1 представлена архитектура PAPI.

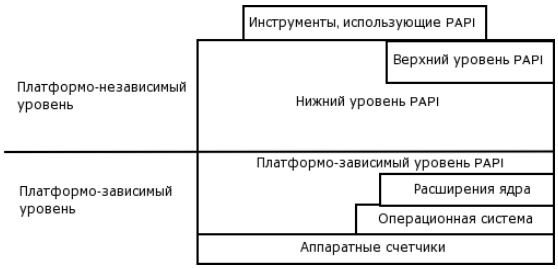

Рис. 1. Архитектура PAPI

Существует два основных уровня PAPI: платформо-независимый и платформо-зависимый, которые скрывают от пользователя детали реализации доступа к аппаратным счетчикам конкретного процессора. Для этого функции PAPI, специфичные для платформы, могут использовать расширения ядра, функции целевой операционной системы или язык ассемблера.

3. Интерфейсы PAPI

PAPI предоставляет пользователям интерфейсы верхнего и нижнего уровня, в которых различается сложность настройки и использования. В настоящее время существуют реализации интерфейса на языках высокого уровня, таких как C и Fortran.

3.1. Интерфейс высокого уровня

Интерфейс высокого уровня состоит всего из 7 функций, обеспечивающих основные операции над аппаратными счетчиками: запуск, остановка, чтение со сбросом и без сброса. В этом случае пользователь может использовать только события, предопределенные стандартом PAPI. Функции интерфейса верхнего уровня используют интерфейс PAPI низкого уровня, чтобы избавить пользователей от явных вызовов, таких как функции инициализации библиотеки PAPI.

```
int PAPI_num_counters (void)
```

Инициализирует PAPI (если требуется). Возвращает число аппаратных счетчиков.

```
int PAPI_start_counters
  (int *events, int len)
```

Инициализирует PAPI (если требуется). Связывает множество событий с аппаратными счетчиками. Запускает счетчики.

```
int PAPI_stop_counters
  (long long *vals, int  alen)
```

Останавливает счетчики и сохраняет их значения в массиве vals.

```
int PAPI_accum_counters
  (long long *vals, int  alen)
```

Прибавляет значения счетчиков к значениям в массиве vals и обнуляет счетчики.

```
int PAPI_read_counters
  (long long *vals, int  alen)
```

Считывает значения счетчиков в массив vals и обнуляет счетчики.

```
int PAPI_flips
  (float *real_time,
   float *proc_time,
   long long *flpins,
   float *mflpins)
int PAPI_flops
  (float *real_time,
   float *proc_time,
   long long *flpins,
   float *mflpins)
int PAPI_ipc
  (float *real_time,
   float *proc_time,
   long long *ins,
   float *ipc)
```

Упрощенные вызовы для измерения числа команд и операций с плавающей точкой, а также частоты выполнения команд процессора. Кроме того, эти функции возвращают реальное время работы процессора, а также виртуальное время, то есть время выполнения пользовательского процесса.

Ниже приведен пример, иллюстрирующий использование функций высокоуровневого интерфейса PAPI. В нем происходит подсчет общего числа команд и тактов процессора при выполнении функции do_work.

```
#include <papi.h>

#define NUM_EVENTS 2

long long values[NUM_EVENTS];
unsigned int Events[NUM_EVENTS] =
{PAPI_TOT_INS, PAPI_TOT_CYC};

/* Стартовать счетчики */
PAPI_start_counters ((int*)Events,
NUM_EVENTS);

/* Интересующая нас функция */
do_work ();

/* Остановить счетчики и сохранить
их значения в массиве values */
ret = PAPI_stop_counters (values,
NUM_EVENTS);
```

3.2. Интерфейс низкого уровня

Низкоуровневый интерфейс обладает по сравнению с интерфейсом высокого уровня расширенной функциональностью и большей эффективностью. В его состав входит более 50 различных функций, которые можно разделить на следующие группы:

- инициализация библиотеки PAPI;
- функции измерения времени;
- функции получения информации;
- служебные функции;
- функции управления множествами событий;
- функции управления аппаратными счетчиками.

Далее показано использование низкоуровневого интерфейса PAPI для подсчета общего числа тактов процессора, а также числа команд сопроцессора плавающей арифметики во время вызова функции do_work.

```
#include <papi.h>

#define NUM_EVENTS 2

int Events[NUM_EVENTS] =
{PAPI_FP_INS,PAPI_TOT_CYC};
int EventSet;
long long values[NUM_EVENTS];

/* Инициализация PAPI */
ret = PAPI_library_init
          (PAPI_VER_CURRENT);

/* Создать множество событий */
ret = PAPI_create_eventset
          (&EventSet);

/* Добавить новые события */
```

```
ret = PAPI_add_events
          (&EventSet,
           Events,
           NUM_EVENTS);

/* Стартовать счетчики */
ret = PAPI_start (EventSet);

do_work(); /* Искомая функция */

/* Остановить счетчики и сохранить
результат в массиве values */
ret = PAPI_stop (EventSet,values);
```

В проекте PAPI предложено стандартизованное, переносимое решение для профилирования кода посредством управления аппаратными счетчиками событий. На сегодняшний день существуют реализации PAPI в виде библиотек для многих современных платформ. Следует отметить также хорошую документированность проекта и простоту использования предлагаемых интерфейсов.

Для используемой целевой системы авторами была доработана и портирована версия papi-c 3.9.0. Такой выбор объясняется слабой зависимостью этой версии от системных вызовов современных ОС семейства Windows или Linux.

4. Потенциальные угрозы, которые можно выявлять с помощью аппаратных счетчиков

4.1. Атаки повторного использования кода

Атаки повторного использования кода (Code Reuse Attacks, CRA [3]), ставящие под угрозу целостность потока управления программы, направлены на изменение нормального потока управления для выполнения вредоносных действий. Среди примеров можно привести возвратно-ориентированное программирование, (Return Oriented Programming, ROP), используя методы которого, злоумышленник получает контроль над стеком вызовов, чтобы заменить адрес возврата из функции. Другим примером является переходо-ориентированное программирование (Jump Oriented Programming, JOP), в котором злоумышленник использует команды перехода для объединения фрагментов вредоносного кода.

Собранная с помощью аппаратных счетчиков информация, такая как, например, события промаха в кэш-памяти или неправильные предсказания ветвления, является, на наш взгляд, хорошим эвристическим индикатором атак этого типа.

4.2. Внедрение кода

Атаки на внедрение кода (Code Injection [4]) реализуют вставки в атакуемое приложение вредоносного кода. Многие из атак этого типа выполняются с помощью переполнения буфера и могут быть выполнены различными способами. В некоторых случаях, для изменения поведения программы могут быть введены ложные данные, такие как ложные показания датчиков в системах управления технологическим процессом. Целями таких атак могут быть захват контроля, саботаж или повреждение атакуемой системы таким образом, чтобы помешать выполнению ее миссии.

Для противодействия этим атакам можно использовать эталонные профили выполнения программы, построенные на аппаратных событиях во время ее первых «эталонных» запусков. В дальнейшем аппаратные счетчики можно использовать для выявления аномального поведения программы, т.е. отклонения текущего профиля выполнения от эталонного. В зависимости от внедренного кода количество различных аппаратных событий, а также их соотношений (как будет показано ниже) резко изменяется за короткое время, что может служит индикатором такого рода атак.

4.3. Атаки по сторонним каналам

Атаки этого типа обычно направлены на кражу информации из целевой системы. Это могут быть пароли, ключи или другие секретные данные.

Для извлечения нужной информации в настоящее время злоумышленники все чаще прибегают к атакам с использованием кэш-памяти: Flush+Reload, Evict+Time, Prime+Probe, Evict+Reload [1].

Все вышеперечисленные атаки порождают определенные аппаратные события, такие, как, например, промахи в кэш-памяти, что является хорошим индикатором для их раннего обнаружения.

4.4. Атаки типа «отказ в обслуживании»

Некоторые атаки типа «отказ в обслуживании» (Denial of Service, DoS) также могут быть обнаружены с помощью аппаратных счетчиков. Как и в п. 4.2 здесь следует использовать эталонные профили выполнения программы, поскольку предполагается, что число, последовательность возникновения и определенные соотношения событий при нормальной работе приложения сильно отличаются от работы во время DoS атаки, которая обычно характеризуется чрезвычайно высокой аппаратной активностью,

в частности должно серьезно возрасти число таких событий, как промахи в TLB и запись в кэш-память первого уровня [5].

5. Применение аппаратных счетчиков в обеспечении информационной безопасности

В качестве экспериментальной платформы был выбран процессор Intel Core I3-6100, под управлением ОС Fedora Core 22. Для имитации атаки использовался инструмент Mastik [6]. Поскольку атаки по сторонним каналам с использованием кэш-памяти предполагают увеличение числа промахов по кэш-памяти 3-го уровня (L3), то вполне логично использовать это событие (L3_MISS) в качестве индикатора потенциальной угрозы. Однако одного его недостаточно. Сама логика приложения может предполагать большое число событий L3_MISS, например, при работе с большим числом данных, не хранящихся локально. Поэтому было предложено еще одно событие (L1_REPL), показывающее, как часто замещаются строки в кэш-памяти 1-го уровня. Теперь если взять их отношение L3_MISS / L1_REPL, то полученный индикатор будет означать, что приложение значительно использует память, но при этом часто очищает кэш. Что может свидетельствовать о проводимой атаке.

Последующие эксперименты показали, что за время измерения атакуемые приложения имели в несколько раз более высокое значение предложенного индикатора (в среднем, в 5 раз), по сравнению с его же значением в обычном режиме работы.

Также были построены профили типичного выполнения вычислительных задач. В качестве критерия было предложено использовать отношение числа выполненных команд сопроцессора плавающей арифметики к числу всех выполненных команд (профиль строился для каждого критического потока управления). Замеры проводились в определенных заранее контрольных точках. Эксперименты показали, что разброс в значениях критерия при разных запусках не превысил 3%. Такой подход позволяет устранить проблему недетерминизма аппаратных счетчиков [7]. В профиль были включены и события по выполнению перехода и выполнению инструкции ветвления. Учитывая недерминизм аппаратных счетчиков, сравнение профилей выполнения задачи в эксплуатационном режиме с построенными в ходе настройки эталонными профилями проводилось по контрольным событиям перехода и ветвления (были выделены контрольные последовательности событий, нарушение которых является признаком потенциального сбоя или атаки). Стресс-тестирование показало, что предложенный подход позволил успешно выявлять некорректные последовательности событий.

6. Заключение

Был проведен анализ потенциальных угроз, реализован прикладной программный интерфейс доступа к аппаратным счетчикам производительности, проведены исследования, подтверждающие изначальное предположение о том, что профиль выполнения атакуемой системы, построенный на определенных аппаратных событиях, меняется, что позволяет построить эффективную систему мониторинга и защиты.

В качестве направления дальнейших исследований видится расширение списка индикаторов, позволяющих выявлять проникновение в систему программ-злоумышленников, путем анализа изменения количества аппаратных событий в их различных комбинациях, во время имитации атак по сторонним каналам.

«Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).»

Using Hardware Performance Counters to Ensure Information Security of Critical Systems

Vladimir Galatenko, Konstantin Kostiukhin

Abstract. The article discusses the possibility of using hardware performance counters, commonly used in the creating of system execution profiles, to identify potential security threats to critically important systems and complexes. The authors have ported an open Performance Application Programming Interface (PAPI), which is used to manage hardware counters.

Keywords: performance counters, information security, side-channels attacks, PAPI

Литература

1. F. Liu, Y. Yarom, Q. Ge, G. Heiser, R.B. Lee. Last-Level Cache Side-Channel Attacks are Practical, Security Privacy. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015.

2. PAPI User's Guide, http://icl.cs.utk.edu/papi/

3. Vishnyakov A.V., Nurmukhametov A.R., Kurmangaleev S.F., Gaisaryan S.S. Method for analysis of code-reuse attacks. Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS). 2018;30(5):31-54. (In Russ.)

4. Oliver Moradov. Code Injection in Brief: Types, Examples, and Mitigation, 2022, https://bright-sec.com/blog/code-injection/

5. Pablo Pessoa do Nascimento, Paulo Pereira, Jr Marco Mialaret, Isac Ferreira, Paulo Maciel. A methodology for selecting hardware performance counters for supporting non-intrusive diagnostic of flood DDoS attacks on web servers, Computers & Security, Volume 110, 2021, https://www.sciencedirect.com/science/article/pii/S0167404821002583

6. Mastik: A Micro-Architectural Side-Channel Toolkit, https://github.com/0xADE1A1DE/Mastik

7. S. Das, J. Werner, M. Antonakakis, M. Polychronakis, F. Monrose. SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security. 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019, pp. 20-38.

Низкоуровневые криптографические операции

Н. Д. Байков[1], А. Н. Годунов[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nknikita@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nkag@niisi.ras.ru

Аннотация. Целью работы является обзор базовых низкоуровневых криптографических операций, лежащих в основе современных криптографических протоколов. Рассмотрены примеры широко применяемых операций криптографического хэширования, шифрования и формирования электронно-цифровой подписи.

Ключевые слова: криптографическая хэш-функция, шифрование, цифровая подпись

1. Введение

Архитектура безопасности большого числа современных систем передачи и хранения информации строится на трех главных видах криптографических операций:

- криптографическое хэширование данных (свертка данных);
- шифрование данных;
- формирование цифровой подписи.

За последние несколько десятилетий было разработано обширное множество различных подходов к реализации этих операций, каждый со своими собственными особенностями, отражающими веяния современных тенденций в криптографии и покрывающими все новые и новые обнаруживаемые уязвимости. Подобное многообразие особенностей и подходов на ранних этапах ознакомления с предметной областью существенно повышает порог входа для новичков, целью которых является внедрение систем безопасности в собственные программные продукты и системы.

Задачей настоящей работы является ознакомление с фундаментальными идеями, лежащими в основе алгоритмов выполнения криптографических операций, и областью применения каждой из этих операций. Изложение преимущественно фокусируется на примерах классических алгоритмов, чтобы сформировать у читателя общее представление об их различиях и предоставить ему возможность самостоятельно определиться с направлением дальнейшего изучения.

Раздел 2 посвящен описанию области применения операции криптографического хэширования данных. В частности, рассмотрено применение алгоритмов в задаче хранения аутентификационных данных (паролей). Сформулированы основные требования к алгоритмам криптографического хэширования, обеспечивающие им свойство криптографической стойкости, т.е. способности алгоритма противостоять атакам на основе методов криптографического анализа. Стойким в данном контексте считается, алгоритм, атака на который требует настолько значительных (или вовсе недостижимых) ресурсов атакующего, что затраты на них значительно превосходят ценность защищаемых данных. Приведены примеры зарубежных алгоритмов бесключевого хэширования, включающие в себя семейства алгоритмов MD [1], SHA [2], [3] и отечественного алгоритма Стрибог [4], [5].

В разделе 3 рассмотрены алгоритмы шифрования и их использование для обеспечения конфиденциальности данных при их передаче (например, по протоколу HTTP с поддержкой Transport Layer Security (TLS) [6]). Описаны основные идеи семейств алгоритмов симметричного шифрования на основе общего секретного ключа и алгоритмов асимметричного шифрования на основе пары из публичного ключа для шифрования данных и секретного ключа для их расшифровки. На примерах алгоритмов симметричного шифрования также продемонстрированы различия между шифрованием данных блоками и поточным шифрованием, описаны достоинства и недостатки этих подходов.

В разделе 4 приведено описание операции формирования цифровой подписи и ее применение для обеспечения гарантий целостности передаваемых данных. Рассмотрены особенности протоколов использования цифровых подписей для проверки подлинности получаемых по сети публичных ключей при установлении зашифрованного соединения с сервером посредством асимметричного алгоритма шифрования.

В разделе 5 рассматриваются примеры готовых программных реализаций всех трех видов операций в составе библиотек и утилит. Примеры включают в себя как зарубежные программные продукты (такие как OpenSSL [7], NaCl [8] или утилиту Crypt4GH, реализующую

стандарт GA4GH [9]), так и отечественные средства криптографической защиты (например, криптопровайдер КриптоПро CSP [10]).

2. Криптографическое хэширование данных

Под хэшированием в широком смысле подразумевается любой односторонний процесс преобразования входных данных в выходные по некоторому заданному алгоритму. Дополнительные требования к алгоритму зависят от области его применения. В криптографии операция хэширования обычно применяется для обеспечения целостности передаваемых данных, а также для повышения безопасности при хранении аутентификационных данных. Для этого алгоритму хэширования требуется обладать дополнительным свойством — свойством криптографической стойкости, которое на практике достигается за счет выполнения следующих основных требований:

- при применении алгоритма к одним и тем же входным значениям всегда должны получаться одни и те же выходные значения (не обязательно уникальные);
- восстановление возможного прообраза по полученным выходным значениям должно иметь крайне высокую трудоемкость и требовать очень большого времени;
- малейшее изменение входных значений должно приводить к существенному и трудно предсказуемому изменению получаемых в результате выходных значений;
- алгоритм должен минимизировать вероятность возникновения коллизий, когда для двух различных входных значений в результате получаются совпадающие выходные значения;
- выходные данные не должны нести в себе какого-либо дополнительного смысла, кроме того, что получение двух различных выходных значений свидетельствует о несовпадении исходных входных значений.

Рассмотрим использование операций хэширования для обеспечения безопасного хранения аутентификационных данных. Общепринятым способом аутентификации пользователя является использование паролей. В современных информационных системах хранение паролей учетных записей пользователей в открытом виде считается нежелательной практикой, т.к. в случае утечки базы данных злоумышленнику не потребуется прикладывать дополнительные усилия для получения доступа к системе. Криптографическое хэширование призвано усложнить злоумышленнику задачу подбора пароля и создать дополнительный запас времени, за который можно было бы успеть обнаружить утечку и изменить пароль до того, как он станет известен злоумышленнику и будет нанесен ущерб системе или пользователю.

Для этой цели значения паролей в базе данных информационной системы заменяются значениями их криптографических хэшей. При этом по-прежнему возможно достаточно надежно аутентифицировать пользователя при входе в систему по значению его пароля: нужно вычислить хэш пароля, а затем сравнить полученный хэш со значением хэша указанного пользователя в базе. Вместе с этим подобная замена значительно понижает риск мгновенной компрометации учетных записей пользователей в случае утечки базы, т.к. для подбора исходного пароля злоумышленнику необходимо решить задачу восстановления прообраза хэша, что сделать достаточно трудно.

На практике только этих мер для обеспечения безопасности оказывается недостаточно. Хотя хэширование само по себе является эффективным средством для сокрытия значения исходного пароля пользователя, оно не учитывает влияние человеческого фактора. Часто пользователи используют в качестве пароля типовые легко запоминающиеся комбинации символов. Если злоумышленнику известен используемый в системе алгоритм криптографического хэширования, он может составить список из наиболее часто встречающихся паролей, заблаговременно вычислить значения всех их хэшей и упорядочить полученную таблицу хэшей для быстрого поиска по ней. Тогда, если произойдет утечка базы, ему будет достаточно сопоставить аутентификационные данные системы с имеющимся у него упорядоченным перечнем хэшей, чтобы в кратчайшие сроки выявить возможные учетные записи со слабыми паролями и получить доступ к системе. Хотя ответственность за использование сложного пароля прежде всего лежит на самом пользователе, тем не менее в качестве дополнительной меры противодействия данной угрозе был предложен метод автоматического усложнения пользовательских паролей, позволяющий еще больше замедлить злоумышленника. Так появилось понятие «соли» (salt) — последовательности символов, добавляемой к паролю при каждом вычислении хэша. Соль случайным образом формируется в момент регистрации пользователя в системе. Сформированные значения сохраняются в базе данных системы в открытом виде, но держатся в секрете от любых внешних участников информационного обмена. Единственная задача соли состоит в том, чтобы за счет ее добавления (конкатенации) усилить пароль и превратить его в более длинную и нечитаемую строку, которой не окажется в заранее

подготовленном и оптимизированном для быстрого поиска перечне злоумышленника. При этом следует понимать, что утечка значений соли вместе с другими аутентификационными данными по-прежнему ставит под угрозу безопасность системы. Отличие только в том, что злоумышленнику понадобится больше времени на взлом, т.к. ему будет необходимо повторно перебрать все типовые пароли и вычислить их хэши с учетом ставшей известной ему соли.

Классическими примерами семейств (бесключевых) криптографических хэш-функций являются семейства MD-функций (MD2, MD4, MD5 [1] и MD6), а также семейство функций SHA (Secure Hash Algorithm). Часть из указанных хэш-функций за прошедшее с момента изобретения время уже утратили свойства криптографической стойкости, тогда как оставшиеся функции до сих пор активно применяются на практике. К последним можно отнести разновидности SHA-2 [2] для различных длин хэшей, а также относительно новый алгоритм SHA-3 (Keccak) [3]. Отечественным вариантом криптографического хэширования выступает алгоритм «Стрибог», описанный в межгосударственном криптографическом стандарте ГОСТ 34.11-2018 [4], разработанном на основе национального стандарта Российской Федерации ГОСТ Р 34.11.2012 [5].

3. Шифрование данных

Другой важной с точки зрения безопасности операцией является шифрование. Ее основное назначение — соблюдение условия конфиденциальности при передаче информации от отправителя к получателю. Требуется, чтобы никакой сторонний субъект, которому удастся перехватить сообщение, не был способен прочесть скрытую в сообщении информацию. Для достижения поставленной цели также используется алгоритм преобразования исходных данных, но — в отличие от алгоритма криптографического хэширования — задача восстановления прообраза зашифрованных данных получателем должна быть однозначно разрешима. Конфиденциальность же обеспечивается тем условием, что для расшифровки данных алгоритму необходим дополнительный аргумент — персональный ключ шифрования, который получатель должен хранить в тайне от других участников информационного обмена и для которого алгоритм шифрования должен гарантировать недопустимо высокую трудоемкость его подбора по структуре зашифрованных данных. Отправителю для шифрования данных также необходим парный к ключу получателя ключ, который в зависимости от используемого алгоритма может быть как секретным, так и публично доступным. Принято выделять два основных семейства криптографических алгоритмов шифрования:

- алгоритмы на основе секретного симметричного ключа, используемого одновременно для шифрования и расшифровки данных;

- алгоритмы асимметричного шифрования с отдельным публичным ключом для шифрования данных и отдельным секретным ключом для расшифровки данных.

Простейшим примером алгоритма симметричного шифрования является предложенный в 1917 году шифр Вернама, предназначавшийся для защищенной передачи телеграфных сообщений. Алгоритм состоит всего лишь из одного действия — применения логической операции XOR. Аргументами операции выступают исходное шифруемое сообщение и секретный ключ, длина которого должна совпадать с длиной сообщения. Для расшифровки данных принимающей стороне также достаточно всего лишь применить операцию XOR при условии, что ей известен секретный ключ. Данный алгоритм, как было показано Шенноном, обладает абсолютной криптографической стойкостью. Однако для его применения на практике необходимо, чтобы в каждой операции всегда был задействован новый случайно генерируемый ключ, длина которого бы совпадала и длиной шифруемых данных. В противном случае алгоритм становится уязвим к методам статистического анализа, т.к. XOR между двумя зашифрованными одним и тем же ключом сообщениями позволяет исключить из рассмотрения значение ключа и определить позиции совпадающих и несовпадающих битов исходных сообщений. Если шифр используется для передачи текстовых данных, поиск совпадений длиной в один или несколько байтов позволит определить позиции совпадающих символов, на место которых можно попытаться подставить наиболее распространенные символы используемого алфавита. Также злоумышленник получит возможность отслеживать отправку повторяющихся сообщений, а взлом шифра хотя бы одного из сообщений сразу же позволит расшифровать все оставшиеся сообщения.

Практическая реализация формирования и безопасной передачи очень длинных и при этом абсолютно случайных ключей весьма затратна. По этой причине в чистом виде шифр Вернама практически не используется. Вместо этого алгоритмы шифрования обычно прибегают к псевдослучайному формированию шифрующей последовательности. Упрощенно это выглядит следующим образом:

- выбирается секретный симметричный ключ

шифрования фиксированной длины;

- задается функция криптографического хэширования, для которой секретный ключ должен являться дополнительным параметром;

- задается последовательность целых неотрицательных чисел *nonce*, *nonce*+1, *nonce*+2, … Точка отсчета *nonce* выбирается таким образом, чтобы среди элементов последовательности не встречались уже ранее использовавшиеся в других операциях шифрования элементы. Как альтернатива также встречается использование фиксированной последовательности 0, 1, 2, 3… с использованием *nonce* в роли одноразового дополнительного параметра для инициализации начального состояния функции хэширования наравне с секретным ключом;

- используя секретный симметричный ключ, путем вычисления хэшей от *nonce*+*i* формируется шифрующая последовательность;

- преобразование исходных данных в шифр осуществляется через XOR с шифрующей последовательностью;

- принимающей стороне зашифрованные данные передаются вместе со значением *nonce*, с помощью которого она должна восстановить шифрующую последовательность из собственного значения секретного ключа.

Описанную выше процедуру принято называть поточным шифрованием, т.к. она позволяет шифровать исходные данные произвольной длины побитово и свободно перемещаться между участками данных при шифровании и расшифровке. Примерами используемых на практике алгоритмов поточного симметричного шифрования являются Salsa20 [11], XSalsa20 (расширенный вариант Salsa20 с увеличением размера *nonce* с 64 до 192 бит), а также ChaCha20 [12].

Альтернативой алгоритмам поточного шифрования выступают алгоритмы блочного шифрования. В них входные данные передаются функции шифрования порциями в виде отдельных блоков фиксированной длины. Широко известными примерами алгоритмов блочного шифрования являются ныне устаревший алгоритм DES (Data Encryption Standard) с длиной ключа 56 бит и пришедший ему на смену AES (Advanced Encryption Standard) с поддерживаемыми длинами ключей 128/192/256 бит [13]. В частности, AES использует блоки размера 128 бит, из которых функция шифрования сначала формирует квадрат 4x4 байта для того, чтобы далее выполнять на нем различные операции перестановки.

Случаи, когда длина передаваемого сообщения точно совпадает с длиной одного блока, на практике встречаются достаточно редко. Гораздо чаще пересылаемые данные имеют произвольную длину, значительно превышающую размеры блока. Простейшим решением в таких случаях было бы дополнение исходного сообщения служебными битами до длины, кратной длине блока, а затем последовательное применение функции шифрования к каждому блоку сообщения. Такой режим использования алгоритма шифрования AES получил название Electronic Codebook (ECB). В реальных системах AES-ECB почти не встречается из-за заложенного в него недостатка — низкой эффективности при работе с данными, имеющими периодическую структуру и содержащими большое количество повторяющихся блоков. Проблемой является то, что для одинаковых входных блоков функция шифрования сформирует одинаковый шифр. Этим она частично раскроет для сторонних наблюдателей структуру исходных данных. Особенно наглядно эту проблему демонстрируют примеры использования AES-ECB для зашифрованной передачи изображений, где через шифр могут визуально угадываться контуры исходного изображения. Для борьбы с данным недостатком блочных алгоритмов были разработаны иные режимы функционирования [14], наиболее известными из которых являются:

- Cipher Block Changing (CBC) или режим сцепления блоков шифротекста;

- Counter Mode (CTR) или режим счетчика;

- Galois/Counter Mode (GCM).

В случае CBC проблема повторяющихся блоков решается добавлением к шифруемым данным «шумов». Алгоритм добавления в этом случае выглядит следующим образом:

- перед началом работы случайным образом выбирается вектор инициализации (IV). Вектор должен иметь длину одного блока и задает начальное значение для шума. Получателю IV передается в открытом виде;

- блоки передаваемого сообщения шифруются последовательно;

- шаг алгоритма состоит из получения очередного блока исходного текста, добавления к нему текущего значения шума с помощью операции XOR и передачи полученного результата на вход функции шифрования блока (AES);

- после каждого шифрования полученный в результате зашифрованный блок устанавливается как новое текущее значение шума.

Хотя данный режим эффективно решает проблему шифрования повторяющихся блоков, у него есть собственные недостатки. Среди них невозможность распараллеливания операции шифрования (хотя при этом присутствует возможность параллельной расшифровки данных), а также наличие угроз вида Padding Oracle Attack [15]. Последнюю рассмотрим более подробно.

Для реализации атаки Padding Oracle Attack достаточно следующих условий:

- злоумышленник способен перехватывать зашифрованные сообщения отправителя;

- злоумышленнику известна используемая длина блока (для AES это 16 байт), поэтому он может разделить сообщение на отдельные блоки и расшифровывать их по отдельности друг от друга;

- злоумышленнику известен используемый при шифровании формат дополнения сообщения служебными байтами до длины, кратной длине блока. Как правило, используется формат, когда к последнему блоку длины 15 байт добавляется один байт со значением 1, для блока длины 14 байт — два байта со значением 2, для блока длины 13 байт — три байта со значением 3 и т.д. Если длина сообщения кратна 16, используется добавочный блок длины 16, у которого все байты имеют значение 16. Ясно, что в этом случае сообщение, последний блок которого оканчивается, например, двумя байтами со значениями 3 и 2, не будет являться корректным;

- злоумышленник может неограниченное отправлять серверу поддельные сообщения; и

- злоумышленник каким-то образом (например, по разнице во времени отклика сервера на его запросы) способен определить, посчитал ли сервер его сообщение некорректным из-за неправильной последовательности добавочных байтов или нет.

Взлом шифра в этом случае производится поблочно. Блоки взламываются побайтово. В каждом блоке байты взламываются в порядке от конца к началу. Взлом каждого байта выполняется методом перебора. Для взлома последнего байта злоумышленник формирует сообщения, состоящие ровно из двух блоков:

- вторым блоком сообщения является взламываемый блок;

- в первом блоке первые 15 байт задаются случайным образом;

- значения последнего байта перебираются в порядке от 0 до 255, до тех пор, пока не обнаружится, что для какого-то из значений сервер посчитал сообщение корректным.

Корректность сообщения означает, что по результатам применения операции расшифровки второго блока по алгоритму AES (или другому блочному алгоритму шифрования) и вычисления XOR между ним и значением первого блока хвостовые байты полученного в результате блока образуют последовательность, которую сервер интерпретирует как правильную последовательность добавочных байтов. Таковой гарантированно является последовательность из одного последнего байта со значением 1. Для некоторых блоков дополнительно возможны последовательности 2-2, 3-3-3 и т.д., отфильтровать которые можно путем модификации первых 15 байтов первого блока поддельного сообщения.

Информации о том, что последний байт после применения процедуры расшифровки в соответствии с режимом CBC для поддельного сообщения имеет значение 1 достаточно для его взлома в исходном сообщении. Для этого нужно последовательно сделать две операции XOR сначала между единичным байтом и последним подобранным байтом первого блока в подставном сообщении злоумышленника, а затем между получившимся результатом и последним байтом предыдущего блока в исходном сообщении отправителя.

Далее, используя уже полученное поддельное сообщение, необходимо применить аналогичную процедуру подбора предпоследнего байта в первом блоке с целью получить после применения алгоритма расшифровки CBC хвостовую последовательность вида 2-2, затем последовательность 3-3-3 для взлома третьего байта с конца и т.д. Таким образом расшифровывается каждый байт в блоке.

Пример с Padding Oracle Attack наглядно демонстрирует, что достаточно малейшего несовершенства реализации алгоритма (например, диагностируемого несовпадения во времени обработки отдельных ветвей алгоритма), чтобы сделать систему полностью уязвимой

Режим CTR фактически превращает работу с блоками в алгоритм поточного шифрования, где функция шифрования блока используется как функция хэширования:

- перед началом работы выбирается одноразовый целочисленный параметр *nonce*, на основе которого строится последовательность *nonce*, *nonce*+1, *nonce*+2, … по числу блоков исходного сообщения;

- к каждому элементу последовательности *nonce*+*i* применяется функция шифрования для получения очередного блока шифрующей последовательности;

- блоки шифра получаются путем суммирования (XOR) блоков исходного текста передаваемого сообщения с полученными блоками шифрующей последовательности;

- получателю вместе с зашифрованными данными в открытом виде передается значение *nonce*, чтобы он при помощи собственной копии симметричного ключа имел возможность воспроизвести значения шифрующей последовательности и расшифровать сообщения.

Данный режим обладает всеми достоинствами и недостатками алгоритмов поточного шифрования. К ограничениям следует отнести недопустимость повторного использования

nonce для шифрования нескольких блоков данных, т.к. это автоматически позволяет выявить позиции совпадающих и несовпадающих битов зашифрованных сообщений любому, кто перехватит шифр. Поэтому в режиме CTR каждый ключ шифрования может быть использован лишь ограниченное число раз и, если *nonce* выбирается в режиме счетчика, должен быть заменен при переполнении счетчика.

Режим GCM (Galois/Counter Mode) можно считать улучшенной версией режима CTR, добавляющей возможность проверки целостности и аутентификации передаваемых данных. Т.к. GCM основан на CTR, его также можно отнести к семейству алгоритмов поточного шифрования. Среди всех перечисленных режимов GCM является в настоящий момент наиболее используемым. Его распространению поспособствовало обнаружение в CBC вышеупомянутых уязвимостей, из-за чего в конечном итоге CBC был исключен из спецификации Transport Layer Security (TLS) в версии 1.3. Результатом этого стало то, что все присутствующие на сегодняшний день в спецификации TLS 1.3 шифры являются поточными [6].

Вторым важным семейством алгоритмов шифрования являются алгоритмы асимметричного шифрования, также известные как алгоритмы шифрования на основе открытого ключа. Их идея заключается в наличии пары ключей, один из которых является общедоступным и используется для шифрования данных, а второй держится в секрете и используется для расшифровки сообщений. Известнейшим примером алгоритма асимметричного шифрования является алгоритм RSA (аббревиатура от фамилий Rivest, Shamir и Adleman), основанный на высокой вычислительной сложности задачи разложения на множители больших полупростых чисел. Процедура формирования RSA-ключей выглядит следующим образом:

- случайно выбираются два простых числа p и q (длины 1024 бит и более);
- вычисляется модуль $n = pq$;
- вычисляется функция Эйлера модуля $\varphi(n) = (p-1)(q-1)$;
- выбирается публичный показатель степени e как любое число, взаимно простое с $\varphi(n)$. Рекомендуется использовать простые числа с небольшим количеством единичных битов в двоичной записи для ускорения операций возведения в степень. Например, 65537;
- из условия $ed \equiv 1(mod\ \varphi(n))$ вычисляется секретный показатель степени d. Ее существование обеспечивается выполнением предыдущего условия. Для нахождения может использоваться расширенный алгоритм Евклида;
- пара (e, n) назначается публичным ключом;
- пара (d, n) назначается секретным ключом.

Тогда операции шифрования и расшифровки для произвольного числа $0 \le m < n$ задаются следующими симметричными друг другу формулами:

$$c = Enc(m) = m^e\ (mod\ n)$$
$$m = Dec(c) = c^d\ (mod\ n)$$

Обоснование их работоспособности приведено в Приложении к статье. Криптографическая стойкость шифра обеспечивается трудностью вычисления множителей p и q по значению n, а следовательно, и трудностью подбора секретного показателя степени d.

Недостатки RSA схожи с недостатками алгоритмов блочного шифрования — шифр для повторяющихся исходных данных будет одинаков. По этой причине на практике обычно используется комбинированный подход, когда RSA используется на начальном этапе для защищенной передачи симметричного случайно формируемого сеансового ключа, с помощью которого уже осуществляется непосредственное шифрование данных.

Важным достоинством RSA в сравнении с алгоритмами симметричного шифрования является то, что отправителю данных не требуется заранее иметь собственную копию секретного ключа для обмена данными с получателем. Получателю достаточно прислать свой публичный ключ по любому открытому каналу данных. Тогда с его помощью отправитель сможет зашифровать данные для получателя и тем самым обеспечить их конфиденциальность. Однако при этом возникает дополнительная угроза, связанная невозможностью установления реального владельца секретного ключа, от которого был получен публичный ключ. Главной опасностью в этом сценарии являются атаки вида «человек посередине» (Man in the Middle; MITM), в которых злоумышленник:

- формирует собственную пару из публичного и секретного RSA-ключей;
- встраивается в канал между отправителем и получателем данных;
- перехватывает публичный ключ, которым должны шифроваться данные для получателя, при его передаче отправителю;
- подменяет перехваченный ключ своим собственным экземпляром публичного ключа, для которого ему известен секретный ключ.

В таком случае злоумышленник получает возможность расшифровки всех сообщений отправителя, т.к. они шифруются публичным ключом злоумышленника. При этом для отправителя этот факт может остаться незамеченным, т.к. злоумышленник способен самостоятельно перешифровать скомпрометированные данные пере-

хваченным публичным ключом исходного получателя и переслать их дальше. Для борьбы с этим недостатком был разработан метод защиты, основанный на использовании электронно-цифровой подписи (ЭЦП, далее — цифровая подпись), которая должна принадлежать третьей доверенной стороне.

4. Цифровая подпись

Цифровая подпись — это третий вид низкоуровневых криптографических операций, направленный прежде всего на обеспечение защиты целостности передаваемых данных. Выше при описании алгоритма асимметричного шифрования RSA подчеркивалось, что операции шифрования и расшифровки имеют идентичный друг другу вид, из которого следует, что публичный ключ RSA может быть использован не только для шифрования данных, но также с его помощью можно выполнять обратную операцию — расшифровку данных, зашифрованных при помощи секретного ключа. Если передаваемые в открытом виде исходные данные необходимо защитить от искажения или подмены при передаче, можно сделать это следующим образом:

- к передаваемому в открытом виде сообщению прикрепляется дополнительное поле, содержащее хэш-сумму передаваемых данных;
- хэш-сумма дополнительно шифруется секретным ключом владельца данных.

В этом случае любой обладатель публичного ключа при получении данных может самостоятельно вычислить хэш-сумму полученного сообщения, а затем сравнить ее с хэш-суммой, получаемой в результате расшифровки переданного вместе с сообщением дополнительного поля шифротекста. Совпадение значений будет свидетельствовать о том, что сообщение было получено от обладателя секретного ключа и доставлено получателю в неизменном виде.

Однако в случае атаки MITM (Man in the Middle) исходными данными являются сами публичные ключи сервисов сети Интернет, поэтому применить описанный выше алгоритм проверки подписи не представляется возможным и для решения задачи необходимо привлечение третьей стороны. В качестве одного из таких решений было предложено помещать публичные ключи каждого сервиса внутрь специализированного документа — сертификата, дополнительно хранящего информацию о владельце сервиса. При этом право выпуска сертификатов для регистрируемых в сети сервисов предоставляется только ограниченному кругу доверенных организаций — удостоверяющих центров (Certification authority, CA). Для защиты от подделки каждый сертификат заверяется электронной подписью удостоверяющего центра (или подписью нижестоящей в цепочке сертификации организации, сертификат которой заверен подписью удостоверяющего центра). При этом используется описанный выше алгоритм. Публичные ключи корневых удостоверяющих центров, необходимые для проверки подлинности сертификата, общедоступны и, как правило, изначально зашиты в дистрибутивы программ (браузеров), с помощью которых осуществляется доступ к информационным системам в сети Интернет. Таким образом, предустановленные публичные ключи удостоверяющих центров позволяют пользователям программ в автономном режиме проверять целостность публичных ключей внутри получаемых от сервисов сертификатов путем проверки их подписей, которые также должны принадлежать удостоверяющим центрам. Тем самым обеспечивается доверие к отправителю сертификата и исключается возможность подмены публичного ключа на этапе установления защищенного соединения. Подробно структуру сертификата описывает формат X.509 [16].

Детальное описание всех процедур, связанных с использованием алгоритмов асимметричного шифрования, можно найти в серии спецификаций Public-Key Cryptography Standards (PKCS), созданных корпорацией RSA. Среди них:

- PKCS #1 — описание алгоритма RSA и формата его ключей (RFC 8017 [17]);
- PKCS #3 — описание алгоритма Диффи-Хеллмана выработки общего секретного ключа [18];
- PKCS #7 — описание формата зашифрованного и/или подписанного криптографического сообщения (обычно представлен файлом с расширением .p7b) [19];
- PKCS #8 — описание формата секретного ключа (RFC 5958 [20]; обычно представлен файлом с расширением .key);
- PKCS #12 — описание формата экспорта секретного ключа вместе с сертификатом и путём сертификации (RFC 7292 [21]; обычно представлен файлом с расширением .pfx или .p12).

5. Программные реализации

Следует отметить, что даже при наличии готовых криптографических алгоритмов их практическая реализация остается весьма нетривиальной задачей. Как было отмечено в примере с Padding Oracle Attack, для взлома шифра может быть достаточно малейшего неучтенного канала данных. Даже несовпадения времени выполнения операции расшифровки блоков с корректным и некорректным дополнением служебными

хвостовыми байтами оказывается достаточно для проведения тайминговых атак и взлома алгоритма. По этой причине собственная разработка криптографических примитивов при отсутствии должного уровня квалификации часто нежелательна, а при использовании готовых решений следует тщательно подходить к выбору и полагаться только на проверенные реализации алгоритмов, предоставляемые сторонними библиотеками и программными продуктами.

Одним из важнейших примеров готовой реализации является библиотека OpenSSL [7], которая является библиотекой с открытым исходным кодом и известна прежде всего тем, что содержит в себе открытую реализацию протокола TLS. Помимо этого, библиотека поддерживает множество алгоритмов шифрования и криптографического хэширования, включая алгоритмы AES, ChaCha20, RSA и алгоритмы MD5, SHA-2, SHA-3, а также многие другие.

Другим известным примером является библиотека Networking and Cryptography Library (NaCl) [8], созданная Дэниэлом Бернштейном — автором многих известных криптографических алгоритмов. Ядром библиотеки NaCl выступают три основных элемента:

- эллиптическая кривая Curve25519, посредством которой реализуется разновидность протокола Диффи-Хеллмана выработки общего симметричного ключа (ECDH);

- потоковое шифрование на основе алгоритма Salsa20;

- функция Poly1305, используемая для аутентификации сообщений.

При создании библиотеки авторы ставили своей основной целью обеспечение высокой производительности при сохранении небольшого размера библиотеки. Также особое внимание уделено борьбе с тайминговыми атаками по типу описанной Padding Oracle Attack. Для их предотвращения при реализации библиотеки выбирались операции с фиксированным временем выполнения, не зависящим от подаваемых на вход алгоритмам данных.

Среди отечественных криптопровайдеров следует отметить КриптоПро CSP [10], выступающего в роли хранилища ключей и реализующего работу с ними через различные криптографические алгоритмы — хэширование, шифрование и формирование электронных подписей. Достоинством КриптоПро CSP является поддержка как зарубежных, так и отечественные криптографических алгоритмов, а также совместимость с большинством известных разновидностей ключевых носителей. В частности, поддерживаются отечественные алгоритмы криптографического хэширования ГОСТ Р 34.11-2012 (алгоритм «Стрибог»), блочного шифрования ГОСТ Р 34.12-2015 (алгоритмы «Кузнечик» и «Магма») [22] и формирования цифровой подписи ГОСТ Р 34.10-2012 [23].

Другим направлением развития средств криптографии является разработка специализированных библиотек и утилит, направленных на расширение условий применения базовых криптографических операций. Например, во всех в рассмотренных выше подходах к шифрованию неявно предполагалось, что доступ к данным в расшифрованном виде имеют только два участника информационного обмена — владелец данных, выступающий в роли отправителя, а также их получатель, обладающий копией секретного ключа шифрования. При этом шифрование в описанных подходах было прежде всего необходимо для безопасной передачи данных от отправителя к получателю, а вопрос хранения данных отдельно не рассматривался. Часто, когда исходные данные обладают высоким уровнем конфиденциальности, их хранение в открытом виде оказывается нежелательным, поэтому для защиты от утечки данных также применяют шифрование. Если появляется необходимость предоставить дополнительный доступ к зашифрованным данным, существует несколько подходов к реализации этого требования:

- расшифровать исходные данные, чтобы повторно зашифровать их ключом получателя;

- зашифровать ключом получателя ключ, позволяющий расшифровать данные, и передать его вместе с зашифрованными данными (данный подход принято называть envelope encryption).

Развитием второго подхода служит формат Global Alliance for Genomics and Health (GA4GH [9]), разработанный для безопасного хранения и передачи генетических данных пациентов и масштабирующий предложенную концепцию на случай большего количества получателей данных. В GA4GH сообщения имеют блочную структуру и состоят из блоков заголовков (по количеству получателей данных) и блоков зашифрованных данных. Упрощенно формат расшифровки сообщения в GA4GH выглядит следующим образом:

- отправитель и все получатели сообщения имеют собственную пару из публичного и секретного ключей схемы асимметричного шифрования. Публичные ключи получателей предварительно передаются отправителю для шифрования;

- получатель сообщения поочередно перебирает заголовки сообщения в поисках предназначающегося ему заголовка. Каждый заголовок содержит в себе публичный ключ отправителя и блок зашифрованных данных. При обнаружении заголовка, данные которого получится расшиф-

ровать, процедура перебора приостанавливается;

- для расшифровки заголовка необходим симметричный ключ. Для его вычисления используется алгоритм Диффи-Хеллмана выработки общего секретного ключа. На вход алгоритму передаются публичный ключ отправителя из заголовка и секретный ключ получателя (для усиления безопасности в GA4GH значение общего ключа дополнительно хэшируется вместе со значениями публичных ключей по алгоритму Blake2b; см. [9], [24]);

- целостность данных и правильность их расшифровки контролируются с помощью Message Authentication Code (MAC) на основе функции функция Poly1305;

- в случае успеха из заголовка извлекается ключ шифрования данных, с помощью которого расшифровываются блоки передаваемых данных сообщения.

Следует отметить, что получатели данных имеют возможность не только расшифровать данные, но также добавить к сообщению новых получателей, для чего им достаточно знать публичный ключ нового получателя. В этом случае к сообщению добавляется новый блок заголовка, внутрь которого помещается публичный ключ того, кто этот заголовок добавляет.

Одной из известных реализаций формата GA4GH является утилита Crypt4GH на языке программирования Python.

6. Заключение

В работе рассмотрены примеры популярных низкоуровневых криптографических операций. На этих примерах продемонстрированы основные идеи, благодаря которым обеспечивается выполнение требований безопасности.

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).

Приложение

Покажем корректность задания операций шифрования и расшифровки в алгоритме RSA с помощью формул

$$c = Enc(m) = m^e \ (mod\ n)$$
$$m = Dec(c) = c^d \ (mod\ n)$$

Требуется доказать, что для любого $0 \le m < n$ выполнено следующее тождество:

$$Dec\big(Enc(m)\big) = m^{ed} \equiv m \ (mod\ n)$$

Доказательство этого утверждения опирается на использование нескольких вспомогательных теорем.

Теорема 1 (малая теорема Ферма). *Пусть p — простое число и a — натуральное число, которое не делится на p. Тогда*

$$a^{p-1} \equiv 1 \ (mod\ p)$$

Теорема 2 (китайская теорема об остатках). *Пусть $a_1, a_2, \dots, a_n$ — набор попарно взаимно простых натуральных чисел. Пусть также $r_1, r_2, \dots, r_n$ — набор целых чисел таких, что $0 \le r_i \le a_i$ для всех $i \in \{1, 2, \dots, n\}$. Тогда существует такое натуральное N, что для любого $i \in \{1, 2, \dots, n\}$*

$$N \equiv r_i \ (mod\ a_i)$$

При этом, если найдутся два таких числа N_1 и N_2, то

$$N_1 \equiv N_2 \ (mod\ a_1 \cdot a_2 \cdot \dots \cdot a_n)$$

Следствие 1. *Пусть есть два натуральных числа N и M такие,* что *для любого $i \in \{1, 2, \dots, n\}$*

$$N \equiv M \ (mod\ a_i)$$

где $a_1, a_2, \dots, a_n$ — набор попарно взаимно простых натуральных чисел. Тогда

$$N \equiv M \ (mod\ a_1 \cdot a_2 \cdot \dots \cdot a_n)$$

Перейдем к доказательству утверждения. Из условия $ed \equiv 1(mod\ \varphi(n))$ имеем, что для некоторого целого *k* выполнено тождество

$$ed = 1 + k(p-1)(q-1)$$

Тогда

$$m^{ed} = m(m^{p-1})^{k(q-1)}$$

Если *m* не делится на простое число *p*, из малой теоремы Ферма имеем, что

$$m^{p-1} \equiv 1(mod\ p)$$

Отсюда получаем, что

$$m^{ed} \equiv m\ (mod\ p)$$

Если m делится на p, также имеем

$$m^{ed} \equiv 0\ \equiv m\ (mod\ p)$$

Аналогично для простого числа q имеем

$$m^{ed} \equiv m\ (mod\ q)$$

Отсюда по следствию из китайской теоремы об остатках для $n = pq$ получаем

$$m^{ed} \equiv m\ (mod\ n)$$

■

Low Level Cryptographic Operations

N. D. Baykov, A. N. Godunov

Abstract. The goal of this manuscript is to provide an overview of the basic low level cryptographic operations at the heart of modern cryptographic protocols. We consider the examples of some widely used operations of cryptographic hashing, encryption and the digital signature calculation.

Keywords: cryptographic hash function, encryption, digital signature

Литература

1. R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321. 1992. url: https://www.rfc-editor.org/info/rfc1321 (дата обращения 13 октября 2023). doi: 10.17487/RFC1321.

2. T. Hansen. US Secure Hash Algorithms (SHA and HMAC-SHA). RFC 4634. 2006. url: https://datatracker.ietf.org/doc/html/rfc4634 (дата обращения 13 октября 2023). doi: 10.17487/RFC4634.

3. M.J. Dworkin. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Federal Inf. Process. Stds. (NIST FIPS) – 202. 2015. doi: 10.6028/NIST.FIPS.202.

4. ГОСТ 34.11-2018 «Информационная технология. Криптографическая защита информации. Функция хэширования». ФГУП «СТАНДАРТИНФОРМ». Москва. 2018.

5. ГОСТ Р 34.11-2012 «Информационная технология. Криптографическая защита информации. Функция хэширования». ФГУП «СТАНДАРТИНФОРМ». Москва. 2013.

6. E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. 2018. url: https://www.rfc-editor.org/info/rfc8446 (дата обращения 13 октября 2023). doi: 10.17487/RFC8446.

7. OpenSSL. url: https://www.openssl.org (дата обращения 13 октября 2023).

8. NaCl: Networking and Cryptography library. url: http://nacl.cr.yp.to (дата обращения 13 октября 2023).

9. GA4GH File Encryption Standard. 2019. url: http://samtools.github.io/hts-specs/crypt4gh.pdf (дата обращения 13 октября 2023).

10. КриптоПро CSP. url: https://www.cryptopro.ru/products/csp (дата обращения 13 октября 2023).

11. D.J. Bernstein. The Salsa20 Family of Stream Ciphers. New Stream Cipher Designs. Lecture Notes in Computer Science. V 4986 (2008). 84-97. doi: 10.1007/978-3-540-68351-3_8.

12. Y. Nir, A. Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 8439. 2018. url: https://www.rfc-editor.org/info/rfc8439 (дата обращения 13 октября 2023). doi: 10.17487/RFC8439.

13. Advanced Encryption Standard. Federal Inf. Process. Stds. (NIST FIPS) - 197. 2001. doi: 10.6028/NIST.FIPS.197.

14. D. Blazhevski, A. Bojinovski, B. Stojcevska, V. Pachovski. Modes of Operation of the AES Algorithm. “Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)”, India, Mumbai, 2013. url: https://www.researchgate.net/publication/236656798_MODES_OF_OPERATION_OF_THE_AES_ALGORITHM (дата обращения 13 октября 2023).

15. Heaton R. The Padding Oracle Attack. 2013. url: https://robertheaton.com/2013/07/29/padding-oracle-attack/ (дата обращения 13 октября 2023).

16. D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280. 2008. url: https://www.rfc-editor.org/info/rfc5280 (дата обращения 13 октября 2023). doi: 10.17487/RFC5280.

17. K.Ed. Moriarty, B. Kaliski, J. Jonsson, A. Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017. 2016. url: https://www.rfc-editor.org/info/rfc8017 (дата обращения 13 октября 2023). doi: 10.17487/RFC8017.

18. R. Merkle. Secure Communications Over Insecure Channels. Communications of the ACM. V. 21 (1978). № 4. doi:10.1145/359460.359473.

19. R. Housley. Cryptographic Message Syntax (CMS). RFC 5652. 2009. url: https://www.rfc-editor.org/info/rfc5652 (дата обращения 13 октября 2023). doi: 10.17487/RFC5652.

20. S. Turner. Asymmetric Key Packages. RFC 5958. 2010. url: https://www.rfc-editor.org/info/rfc5958 (дата обращения 13 октября 2023). doi: 10.17487/RFC5958.

21. K.Ed. Moriarty, M. Nystrom, S. Parkinson, A. Rusch, M. Scott. PKCS #12: Personal Information Exchange Syntax v1.1. RFC 7292. 2014. url: https://www.rfc-editor.org/info/rfc7292 (дата обращения 13 октября 2023). doi: 10.17487/RFC7292.

22. ГОСТ Р 34.12-2015 «Информационная технология. Криптографическая защита информации. Блочные шифры». ФГУП «СТАНДАРТИНФОРМ». Москва. 2016.

23. ГОСТ Р 34.10-2012 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи». ФГУП «СТАНДАРТИНФОРМ». Москва. 2012.

24. M-J. Ed. Saarinen, J-P. Aumasson. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). RFC 7693. 2015. url: https://www.rfc-editor.org/info/rfc7693 (дата обращения 13 октября 2023). doi: 10.17487/RFC7693.

Введение в разработку и сопровождение систем, реализующих парадигму интернета вещей

В. А. Галатенко[1], К. А. Костюхин[2]

[1]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, galat@niisi.ras.ru;

[2]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, kost@niisi.ras.ru

Аннотация. В статье рассматривается архитектура типичной системы интернета вещей, выделяется набор требований к ее компонентам, и на основе этих требований формулируются требования к средствам разработки и сопровождения систем, реализующих парадигму интернета вещей.

Ключевые слова: интернет вещей, средства разработки, архитектура, требования

1. Введение

Интернет вещей (Internet of Things, IoT) описывает сеть физических объектов — «вещей», — обладающих аппаратными и программными интерфейсами для получения данных и обмена ими с другими устройствами и системами через Интернет. Сложность этих устройств варьируется от обычных бытовых предметов до промышленных механизмов [1].

Ключевой характеристикой IoT является тот факт, что этим устройствам необходимо постоянно взаимодействовать между собой (и, возможно, с пользователем) для передачи, обработки и приема постоянно меняющейся информации.

Можно сказать, что наделение повседневных объектов элементарным интеллектом и коммуникативными навыками дает нам широкий спектр технологических помощников:

- биометрическая одежда, оснащенная датчиками;
- беспилотные аппараты с автоматическим планированием доставки и построением маршрута;
- автоматический мониторинг и управление бытовой техникой в домах (концепция «умного» дома);
- «умные» медицинские приборы;
- системы управления в промышленном секторе и сельском хозяйстве;
- системы управления городской инфраструктурой (концепция «умного» города).

Этот список можно продолжать и дальше.

Реализация такого видения интернета вещей требует, чтобы компьютеры продолжали становиться меньше, «умнее» и поддерживали широкий спектр протоколов связи. В то время как соответствующие модификации аппаратного обеспечения ни у кого не вызывают сомнения, мало кто понимает, что существенных изменений требует также и встроенное программное обеспечение. Это также предъявляет ряд новых требований к средствам разработки систем интернета вещей.

Целью статьи является анализ архитектуры типичной системы интернета вещей и формулировка требований для аппаратного и программного обеспечения таких систем. В свою очередь, из этих требований логично следуют требования к средствам разработки систем интернета вещей, которые также будут сформулированы.

2. Архитектура IoT

На рисунке 1 представлена типичная архитектура системы, реализующей парадигму интернета вещей.

Рис. 1. Архитектура системы IoT

Такая система обычно состоит из компонентов четырех типов:

- устройства, оборудованные датчиками (сенсорами и актуаторами);
- компоненты, обеспечивающие коммуникацию;
- подсистемы обработки данных;

- устройства, предоставляющие пользовательский интерфейс.

2.1. Устройства, оборудованные датчиками

Это могут быть как совсем примитивные устройства, измеряющие температуру окружающей среды, так и достаточно сложные устройства, ведущие непрерывную видеотрансляцию.

Кроме того, устройства могут быть оснащены датчиками, для обеспечения непосредственной коммуникации с конечным пользователем, например, умная колонка обладает микрофоном для приема голосовых команд. В любом случае на первом этапе данные собираются из окружающей среды с помощью соответствующей аппаратуры.

2.2. Компоненты, обеспечивающие коммуникацию

Затем собранные данные отправляются на сервер или в облако. Здесь следует отметить широкий спектр возможных способов и протоколов связи [2]. Некоторые устройства обмениваются данными по беспроводной сети, используя 802.11 (Wi-Fi), Bluetooth, RFID, сети сотовой связи или технологии глобальной сети с низким энергопотреблением (LPWAN), такие как LoRa, SigFox или NB-IoT. Проводная связь подходит для стационарных устройств.

Каждый вариант не является идеальным и имеет компромиссы между энергопотреблением, дальностью действия и шириной полосы пропускания. Выбор наиболее подходящего варианта подключения зависит от конкретного применения системы интернета вещей, но все они выполняют одну и ту же задачу: осуществление коммуникации с сервером, с другими устройствами IoT или с конечным пользователем, которым может быть как человек, так и устройство.

2.3. Обработка данных

Как только данные попадают на сервер, в облако или на другое устройство IoT, начинается этап их обработки.

Это может быть простая проверка, например, того, что показания температуры находятся в пределах допустимого диапазона. Или же использование систем искусственного интеллекта для идентификации объектов в полученном видео (например, злоумышленников в доме).

Если результат обработки данных удовлетворяет заданным критериям, то обычно система просто продолжает работу в штатном режиме. Если же происходит нештатная с точки зрения логики работы системы ситуация, например, температура повысилась до критических значений или в доме появился неидентифицированный объект, то тогда, как правило, результат работы системы поступает конечному пользователю. Отметим, что конечный пользователь также может быть получателем результатов и штатной работы системы, например, пользователь так называемой «умной» колонки.

2.4. Пользовательский интерфейс

Результат обработки данных поступает к конечному пользователю по-разному. Это может быть сделано с помощью оповещения пользователя (электронная почта, текстовое сообщение, уведомление и т.д.). Например, текстовое оповещение, когда в холодильных камерах компании слишком высокая температура.

Кроме того, у пользователя может быть интерфейс, предоставляющий ему доступ к «сырым» необработанным данным. Например, к системе видеонаблюдения, чтобы самому увидеть «злоумышленника» и принять решение, что делать дальше.

Следует иметь ввиду, что пользователь также может иметь возможность выполнить определенные действия и повлиять на систему. Например, он может удаленно регулировать температуру в холодильной камере с помощью приложения на своем телефоне.

Заметим, что некоторые действия могут быть выполнены автоматически. Вместо того чтобы ждать, пока пользователь отрегулирует температуру, система может сделать это автоматически с помощью предопределенных правил. Аналогичным образом, система предупреждения незаконного вторжения может вместе с оповещением пользователя автоматически уведомлять соответствующие органы.

2.5. Протоколы передачи данных

Устройства IoT и конечные пользователи подключаются друг к другу, серверу или облаку через различные каналы связи, такие как Ethernet, Wi-Fi или модем 4G/3G.

Базовым протоколом связи обычно является UDP или TCP IP-протокол. Для простоты разработки и поддержки стандартизации используются такие протоколы, как MQTT, CoAP, XMPP, AMQP [3].

Протокол выбирается с учетом объема и частоты передачи данных, которыми необходимо обмениваться внутри системы IoT.

2.6. Управление данными

Управление данными включает в себя потоковую передачу данных, фильтрацию данных и хранение данных (в случае потери подключения к серверу или облаку).

Главное требование здесь заключается в том, чтобы свести к минимуму задержки и обеспечить корректность данных при передаче.

2.7. Менеджеры подключений

Эти компоненты отвечают за бесперебойное

подключение к серверу, облаку, а также обеспечивают надежность соединения устройств IoT между собой и с конечными пользователями.

2.8. Управляющий сервер IoT

Управляющий сервер должен иметь встроенные интеллектуальные возможности для анализа и принятия решения о том, какие данные следует передавать по сети дальше для обработки, а какие данные можно кэшировать для автономной обработки, чтобы сэкономить затраты на передачу данных и вычислительную мощность основной системы.

3. Основные технологии, используемые в IoT

Ниже приведены некоторые из наиболее часто используемых технологий в IoT.

Метки **RFID** [4] (Radio Frequency IDentification, радиочастотный код) и хранящийся на них EPC (Electronic Product Code, электронный код продукции) уникальный код, позволяющий компаниям отслеживать свою продукцию на протяжении всего жизненного цикла.

NFC [5] (Near Field Communication, связь ближнего действия) используется для обеспечения двустороннего взаимодействия между электронными устройствами, находящимися на расстоянии порядка 10 см друг от друга. Применяется, в основном, в смартфонах для совершения бесконтактных, например, платежных, операций.

Bluetooth используется для коммуникации на небольших расстояниях (примерно 10 м) между компонентами системы IoT. Преимущественно в портативных устройствах.

Z-Wave [6] – это технология радиочастотной связи с низким энергопотреблением. В основном, используется для управления бытовыми приборами посредством передачи простых управляющих команд с малыми временными задержками.

Sigfox [7] – сотовая беспроводная связь большой дальности действия с низкой пропускной способностью. Низкое энергопотребление гарантирует длительную работу удаленных устройств при минимальной зарядке аккумулятора или техническом обслуживании. Применяется в системах удаленного управления, мониторинга и слежения.

LTE-M [8] (Long-Term Evolution for Machine-Type Communications) разработана над стандартными сотовыми протоколами для обеспечения коммуникации устройств IoT между собой. В частности, эта технология должна удовлетворять таким требованиям, как низкая стоимость устройств, связь с большим покрытием, длительное время автономной работы устройств. Применяется, например, при реализации концепции «умных» городов.

NB-IoT [9] (Narrowband IoT) – это технология беспроводной связи, относящаяся к категории глобальных сетей с низким энергопотреблением (low-power wide-area networks, LPWAN), позволяющих подключать устройства, которым требуются небольшие объемы данных, низкая пропускная способность и длительное время автономной работы. Применяется, например, для считывания информации с различных датчиков: бытовых приборов учета, медицинских датчиков и т.д.

Zigbee [10] – беспроводная технология, разработанная как открытый стандарт подключения для создания недорогих беспроводных сетей передачи данных IoT с низким энергопотреблением. Предназначена, в основном, для передачи данных через зашумленные радиочастотные среды. Применяется в системах управления и мониторинга, при реализации концепции «умных» домов.

LoRaWAN [11] (Long Range Wide Area Network) – технология беспроводной связи, также относящаяся к классу LPWAN. Применяется в системах мониторинга, распределенного управления, в частности, при решении логистических задач.

Wi-Fi – это типичный способ коммуникации в локальных, например, домашних, системах IoT. При работе в локальной сети позволяет производить обмен достаточно большими массивами данных.

4. Требования к программному и аппаратному обеспечению систем IoT

Чтобы понять, какие средства разработки использовать для создания систем IoT, необходимо выделить требования к программному и аппаратному обеспечению, на которых строится система IoT. Далее в этом разделе мы сформулируем эти требования и уже на их основе опишем необходимые требования к средствам разработки систем IoT.

4.1. Вычислительная мощность

Управляющий сервер (пул серверов) системы IoT должен обладать серьезной вычислительной мощностью для обеспечения синхронной обработки большого количества входящих и исходящих запросов, работы с БД, а также, возможно, и для обеспечения работы некоторых систем искусственного интеллекта, например, нейросетей.

4.2. Оптимальный код и аппаратная архитектура

Устройства, используемые в системах IoT, обладают достаточно ограниченными ресурсами, как по размеру встроенной памяти, так и по быстродействию. Поэтому создаваемый программный код систем IoT должен иметь возможность оптимизации, как по размеру, так и по быстродействию. Отметим, что и аппаратная архитектура устройства IoT должна оптимально подходить для решения задач, которые выполняются на этом устройстве.

4.3. Широкий спектр поддерживаемых протоколов коммуникации

В системах IoT используется множество способов коммуникации устройств, поэтому, в первую очередь, на аппаратном уровне необходимо обеспечить гибкие возможности поддержки связи, используя, например, технологии из списка, приведенного в разделе 3.

4.4. Быстрая разработка

Поскольку системы IoT стремительно развиваются, их программное обеспечение должно постоянно оставаться актуальным и отвечать часто меняющимся требованиям. Поэтому средства и методы разработки должны быть адаптированы к такой ситуации. Разработка должна быть максимально простой с использованием современных инструментов.

4.5. Кроссплатформенность, переносимость и следование стандартам

С учетом большого количества устройств IoT с разной архитектурой необходимо, чтобы разработанное ПО с минимальными издержками переносилось на различные устройства или же изначально создавалось так, чтобы работать на широком спектре устройств. Достичь этого можно, например, используя интерпретируемые языки в качестве средств расширения функциональности, а также следуя промышленным стандартам в заданной предметной области. Кроме того, для встраиваемых систем и систем с критической миссией существуют собственные стандарты кодирования [12], которым также необходимо следовать.

4.6. Стандартизованные интерфейсы и масштабируемость

Для тех устройств IoT, которые взаимодействуют с конечными пользователями, важно иметь единообразный пользовательский интерфейс. Кроме того, для улучшения масштабируемости системы IoT, необходимо стандартизовать интерфейсы взаимодействия между устройствами.

4.7. Безопасность

Все устройства IoT должны быть защищены от несанкционированного доступа и устойчивы к взлому. При обеспечении информационной безопасности предпочтение следует отдавать аппаратным средствам. Чем ниже уровень, на котором применяются средства противодействия атакам, тем надежнее защищено устройство. В частности, крайне желательно иметь аппаратные средства шифрования.

4.8. Надежность

Все компоненты системы IoT должны быть надежными. Компонент, постоянно нуждающийся в перезагрузке, практически бесполезен. Поэтому и аппаратное, и программное обеспечение, создаваемое для Интернета вещей, должны соответствовать этому требованию. они должны быть надежными. Хотя любой язык и/или инструмент должны быть способны создавать надежное программное обеспечение, в качестве критериев следует рассматривать простоту и полноту тестирования.

4.9. Стабильность

Для компонентов системы IoT характерно свойство, что к моменту выхода с конвейера устройство морально устаревает. Необходимо обеспечить процесс обновления программного обеспечения «на лету», в полевых условиях. При этом устройство должно продолжать стабильно работать. Более того, отсюда проистекает еще одно требование к средствам разработки. При переходе на новые версии инструментов для создания аппаратно-программного обеспечения сами эти инструменты должны продолжать стабильно работать, обеспечивая обратную совместимость с текущим проектом.

5. Заключение

В работе была проведен анализ архитектуры типичной системы интернета вещей. На основе анализа выделены следующие требования.

Требования к аппаратуре.

- Разработка архитектуры аппаратной платформы, исходя из задач IoT, которые будут под ее управлением выполняться.
- Аппаратная поддержка протоколов и технологий связи, используемых в системах IoT.
- Аппаратная поддержка средств информационной безопасности, например, наличие аппаратных средств шифрования.

Требования к средствам разработки ПО.

- Наличие алгоритмов и средств оптимизации, нацеленных на разные способы оптимизации кода, например, по размеру кода или по быстродействию.
- Простая среда разработки (IDE), поддерживающая разработку на уровне исходного

языка, а также учитывающая особенности архитектуры целевой системы.
- Наличие простых средств расширения функциональности «на лету», например, с использованием интерпретируемых языков.
- Следование стандартам как кодирования, так и на используемые средства и языки разработки.
- Следование стандартам в той предметной области, для которой создается система IoT.
- Использование стандартных и безопасных коммуникационных протоколов (например, на уровне вызовов стандартизованных библиотек).
- Использование проверенных сертифицированных средств разработки, регулярные обновления инструментальной системы, особенно обновления системы безопасности.
- Использование систем тестирования, обеспечивающих максимальное тестовое покрытие.
- Обеспечение обратной совместимости средств разработки.

В качестве направления дальнейшей работы видится внедрение принципов контролируемого выполнения [13] при разработке и поддержке систем интернета вещей.

«Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).»

Introduction to the Development and Maintenance of Systems Implementing the Internet of Things Paradigm

Vladimir Galatenko, Konstantin Kostiukhin

Abstract. The article considers the architecture of a typical Internet of Things system, identifies a set of requirements for components of a typical Internet of Things system, and based on these requirements, the requirements for the development and maintenance of systems implementing the Internet of Things paradigm are formulated.

Keywords: IoT, development tools, architecture, requirements

Литература

1. Oracle: What is IoT?, https://www.oracle.com/internet-of-things/what-is-iot/
2. Deshmukh S., Deshmukh C. Study of Internet of Things and development tools and technology, International Journal of Advanced Innovative Technology in Engineering (IJAITE), Vol. 4, Issue 3, 2019, pp. 20-26.
3. 4 Key IoT Protocols – Learn In Great Detail, https://data-flair.training/blogs/iot-protocols/
4. What is RFID? The Beginner's Guide to How RFID Systems Work, https://www.atlasrfidstore.com/rfid-beginners-guide/
5. NFC Specifications, https://nfc-forum.org/build/specifications
6. Z-Wave Specifications, https://z-wavealliance.org/development-resources-overview/specification-for-developers/
7. Sigfox Device Radio Specifications, https://build.sigfox.com/sigfox-device-radio-specifications
8. Long Term Evolution for Machines: LTE-M, https://www.gsma.com/iot/long-term-evolution-machine-type-communication-lte-mtc-cat-m1/
9. Narrowband Internet of Things Whitepaper, https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_application/application_notes/1ma266/1MA266_0e_NB_IoT.pdf
10. Zigbee alliance. – Zigbee Specification, Revision 22 1.0, https://csa-iot.org/wp-content/uploads/2022/01/docs-05-3474-22-0csg-zigbee-specification-1.pdf
11. LoRa alliance, LoRaWAN Specification v1.1, https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-1
12. Motor Industry Software Reliability Association. – MISRA C Coding Standard, https://misra.org.uk/
13. Бетелин В.Б., Галатенко В.А., Костюхин К.А. Основные понятия контролируемого выполнения сложных систем, Информационные технологии, 2013, стр. 1-32.

Платформа для создания стенда полунатурного моделирования на основе ПЛК «Багет»

С. Е. Базаева[1], Я. А. Зотов[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, bazaeva@niisi.msk.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, zotov@niisi.ras.ru

Аннотация. В статье описан проект программной платформы, предназначенной для создания стендов полунатурного моделирования и цифровых двойников АСУ ТП, в состав которых входит ПЛК «Багет». Предлагаемый авторами подход к разработке инструментальных средств построения стендов позволяет обеспечить современный набор функциональных возможностей для моделирования и широкую область применения платформы.

Ключевые слова: программная платформа, стенд полунатурного моделирования, АСУ ТП, ПЛК «Багет»

1. Назначение и область применения программной платформы

Программная платформа для создания стендов полунатурного моделирования и цифровых двойников (ПП ПСиЦД) предназначена для использования на этапах разработки, исследования и эксплуатации автоматизированных систем управления технологическими процессами (АСУ ТП), в состав которых включен программируемый логический контроллер ПЛК «Багет» [1].

Созданный на основе платформы стенд может быть использован для моделирования в реальном времени работы АСУ ТП с целью отладки алгоритмов, для оценки, настройки и исследования параметров АСУ ТП, а также для создания тренажеров, на которых операторы смогут осваивать работу с АСУ ТП.

Полунатурные стенды и цифровые двойники – прекрасный инструмент для отработки действий АСУ ТП в нештатных ситуациях и накапливания данных о функционировании АСУ ТП в целом с целью повышения качества и эффективности управления.

Порядок использования полунатурных стендов и цифровых двойников для проведения массовых испытаний с последующим анализом данных специфицирован в стандартах [2, 3, 4]. Данный способ разработки позволяет достигнуть более точного соответствия АСУ ТП требованиям, указанным в техническом задании, повысить надежность работы АСУ, а, следовательно, и всего объекта, под управлением автоматизированной системы.

Задача платформы ПП ПСиЦД состоит в том, чтобы предоставить разработчикам программного обеспечения АСУ ТП удобный и надежный инструментарий для создания полунатурных стендов и цифровых двойников, соответствующий современным требованиям, предъявляемым к системам разработки. Это позволит обеспечить отладку и исследование штатных алгоритмов управления в реальном масштабе времени.

2. Метод полунатурного моделирования

Метод полунатурного моделирования предполагает частичное использование штатной аппаратуры объекта-оригинала (то есть, моделируемого объекта) в составе стенда, что является основным преимуществом по сравнению с методом математического моделирования, при котором поведение всех компонентов объекта-оригинала должно быть формализовано и вычислено.

Задача формализации поведения технологического процесса целиком сама по себе является нетривиальной и в некоторых случаях, возможно, даже не имеет решения. В ходе построения математической модели сложного многокомпонентного технологического процесса неизбежно происходит упрощение объекта-оригинала, что снижает степень соответствия модели и исходного объекта. Как правило, математическая модель выполняет расчеты, касающиеся содержательной стороны технологического процесса, но не учитывает, например, компоненты, обеспечивающие интерфейс отдельных узлов объекта-оригинала. Также математическая

модель, требующая объемных вычислений, может использовать специальное модельное время, отличающееся от реального времени технологического процесса. Как показывает богатая история развития и применения методов математического моделирования, для широкого круга задач подобное упрощение вполне приемлемо. Однако в случае моделирования систем с жесткими ограничениями на время реакции, когда на счету миллисекунды, затрачиваемые на прохождение данных по каналам связи, более предпочтительным будет включение интерфейсных модулей и даже кабелей штатной длины в состав стенда, чем использование теоретических оценок времени передачи данных.

Поскольку полунатурные стенды и цифровые двойники устроены по одному и тому же принципу, далее будем рассматривать только полунатурные стенды.

3. Состав оборудования, используемого для функционирования ПП ПСиЦД

Начнем с описания АСУ ТП, для моделирования которой предназначена платформа ПП ПСиЦД.

Конфигурация АСУ ТП в одном из вариантов штатного исполнения представлена на рис.1.

Данная система управления включает «бортовой» вычислитель (БЦВМ), исполняющий возложенные на него функции управления технологическим процессом, и ПЛК «Багет» в количестве, необходимом для реализации функций управления технологическим процессом на объекте управления (ОУ). В качестве БЦВМ может быть использована ЭВМ семейства «Багет» или ЭВМ с процессорной архитектурой Intel (ПК).

На ЭВМ семейства «Багет» должна быть установлена операционная система реального времени семейства ОСРВ Багет [5], на ЭВМ с процессорной архитектурой Intel должна быть установлена ОС семейства Linux.

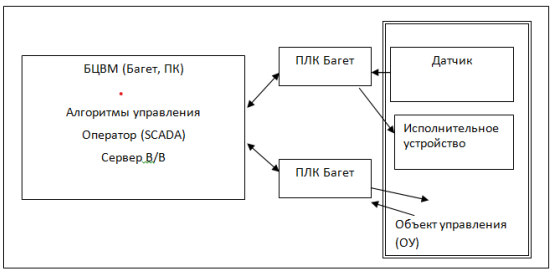

Рис. 1. АСУ ТП на основе ПЛК «Багет» в штатном исполнении

Объект управления обладает датчиками, которые служат источником информации о состоянии объекта, и исполнительными устройствами, с помощью которых АСУ ТП воздействует на ОУ. Датчики и исполнительные устройства подсоединяются к ПЛК в количестве, определяемом конструкцией ПЛК и потребностями технологического процесса.

Функционирование системы контролирует оператор с помощью системы типа SCADA.

Передачу данных между программами, исполняемыми на БЦВМ, и ПЛК обеспечивает сервер ввода-вывода (Сервер В/В).

Использование АСУ ТП в любой области применения подразумевает наличие хранилища данных, необходимых для анализа работы системы. Задача организации сбора и хранения данных не представляет сложности для квалифицированного разработчика и имеет множество решений, поэтому компоненты системы, обеспечивающие данную функциональную возможность, не включены в схему на рис. 1.

Как указано выше, на рис. 1 представлен лишь один из вариантов аппаратно-программной конфигурации. В реальных условиях и в зависимости от решаемой задачи состав и конфигурация используемых компьютеров могут быть иными.

В простейшем случае при небольшой вычислительной нагрузке, с которой может справиться один ПЛК, алгоритмы управления технологическим процессом могут быть реализованы непосредственно на нем. Тогда на БЦВМ функционируют только система SCADA и сервер ввода-вывода, обеспечивающий передачу данных между SCADA и ПЛК. Принципы создания полунатурного стенда в такой конфигурации рассматриваются в [6].

В более сложном случае, при наличии нескольких ПЛК в системе, необходимо будет вынести алгоритмы управления всей конфигурацией на БЦВМ, агрегирующую данные периферии. Программы, исполняемые на ПЛК и обслуживающие отдельные крупные узлы объекта-оригинала, должны быть синхронизированы с программами БЦВМ в реальном времени.

Конфигурация АСУ ТП становится еще более сложной, когда алгоритмы управления выполняются на нескольких ЭВМ, то есть, используют распределенные вычисления.

Однако в данном проекте мы будем ориентироваться на «скромный» вариант, представленный на рис. 1, поскольку, овладев методами отладки программного обеспечения АСУ ТП в указанных условиях, разработчик сможет в дальнейшем масштабировать свою деятельность.

Типовая конфигурация стенда полунатурного моделирования, используемого при функционировании ПП ПСиЦД, представлена на рис. 2.

Данный вариант стенда подразумевает использование штатных компонентов АСУ ТП (окружены пунктирной линией), в интересах которой выполняется моделирование, за исключением «оконечных» устройств (датчиков и исполнительных устройств объекта управления).

Отсутствующий объект-оригинал и входящие в его состав штатные устройства замещаются с помощью моделирующей среды, которая реализует модель объекта управления: принимает данные, направляемые бортовой ЭВМ через ПЛК на исполнительные устройства, изменяет состояние модели и передает данные, имитирующие работу датчиков, на ПЛК и далее на бортовую ЭВМ. Более детально структура моделирующей среды показана на рис. 3.

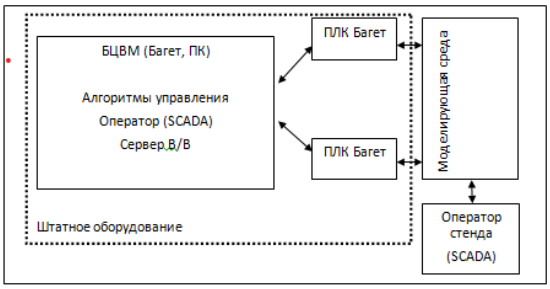

Рис. 2. Стенд полунатурного моделирования, на котором функционирует ПП ПСиЦД

В некоторых случаях реальные «оконечные» устройства могут быть без особого труда включены в состав полунатурного стенда. Например, штатные индикаторные панели, которые не отправляют данные на ПЛК, а только принимают сигналы, могут быть сохранены в составе полунатурного стенда вместе со своими интерфейсами. Также, в зависимости от требований к уровню адекватности и реалистичности моделей, могут быть использованы натурные имиторы «оконечных» устройств и реальные пульты операторов технологического процесса.

Информация о функционировании стенда полунатурного моделирования накапливается в отдельном хранилище (на схеме не показано).

Оператор стенда управляет работой стенда с помощью системы типа SCADA: настраивает параметры и конфигурации, запускает эксперименты, инициирует нештатные ситуации.

Структура стенда, изображенная на рис. 2, не включает средств синхронизации ЭВМ, используемых на стенде, и средств имитации нештатных ситуаций. Более подробно эти вопросы рассматриваются в последующих разделах.

Детализированная структура моделирующей среды показана на рис. 3.

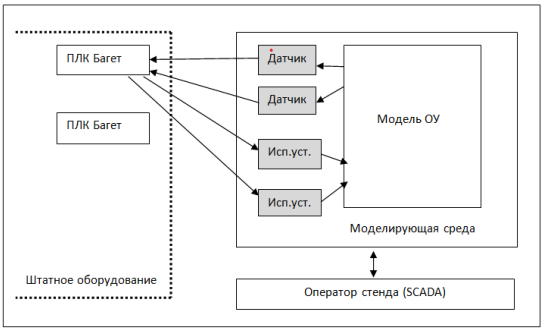

Рис. 3. Структура моделирующей среды в составе ПП ПСиЦД

Моделирующая среда реализуется на одной или нескольких ЭВМ (например, на ЭВМ с процессорной архитектурой Intel), объединенных локальной сетью. Программное обеспечение моделирующей среды (Модель ОУ) включает алгоритмы моделирования объекта-оригинала, которым управляет АСУ ТП.

Закрашенные серым цветом клетки включены в рисунок для пояснения структурных связей на стенде и обозначают датчики и исполнительные устройства ОУ, в реальности отсутствующие на стенде.

4. Состав программного обеспечения и алгоритм функционирования ПП ПСиЦД

4.1. Состав программного обеспечения ПП ПСиЦД

Программное обеспечение ПП ПСиЦД предлагается разрабатывать на языках Си и JavaScript. Эти языки программирования широко используются на различных архитектурах, что обеспечит мобильность ПО и облегчит сопряжение с имеющимися разработками. Язык JavaScript также позволяет формировать спецификации в виде текстовых файлов, что может быть удобно для инженерных работников, в компетенции которых не входит умение программировать на языке Си.

В состав программного обеспечения ПП ПСиЦД входят следующие компоненты:

– среда разработки;
– среда выполнения;
– подсистема управления выполнением;
– подсистема хранения данных по экспериментам;
– подсистема анализа данных моделирования.

Основой для построения алгоритма функционирования ПП ПСиЦД являются:

– штатное программное обеспечение АСУ ТП;
– набор программ, моделирующих поведение

реального оборудования, которое не представлено на стенде;
- программы и библиотеки функций, обеспечивающие взаимодействие реальной аппаратуры и ПО с моделирующей средой.

Платформа ПП ПСиЦД работает в следующих режимах:
- режим разработки и отладки моделей (работает программист-разработчик);
- режим конфигурирования и настройки работы стенда (работает программист-разработчик или инженер-экспериментатор);
- режим выполнения эксперимента по моделированию с использованием готовых конфигураций (работает инженер-экспериментатор или оператор стенда).

Работа с платформой ПП ПСиСД начинается с подготовки модели технологического процесса. При этом необходимо решить две важные задачи: определить правила передачи данных между штатной и моделирующей частями стенда и синхронизировать работу этих частей по времени.

4.2. Дисциплина синхронизации и обмена данными

Предлагается в рамках ПП ПСиЦД применить дисциплину синхронизации и обмена данными, построенную на принципах стандарта High Level Architecture (HLA) [7]. Данная дисциплина определит порядок передачи данных между штатным программным обеспечением, исполняемым на штатной вычислительной аппаратуре, и программным обеспечением моделирующей среды, которое исполняется на вычислительной аппаратуре, предназначенной для реализации процесса моделирования.

Основное требование дисциплины синхронизации и обмена данными состоит в том, что для передачи данных могут быть использованы только библиотечные функции, реализующие данную дисциплину.

Если программист-разработчик, работающий на платформе ПП ПСиЦД, создает (или адаптирует имеющиеся) модели в соответствии с установленной дисциплиной синхронизации и обмена данными, это гарантирует функционирование и совместимость моделирующей среды и штатного ПО независимо от предметной области моделируемой АСУ ТП, поскольку связь между ними экранирована библиотечными функциями и происходит только путем передачи значений параметров по заданному графику.

Обеспечение синхронной работы в реальном времени всех ЭВМ, входящих в состав стенда, является обязательным требованием для любого стенда полунатурного моделирования.

Технологический процесс на практике осуществляется в соответствии с заданным графиком: в установленные моменты времени опрашиваются показания датчиков (например, температура в печи, где выпекается хлеб, и значение таймера). Если температура ниже или выше той, что предусмотрена технологией, поступает сигнал на нагревательные элементы. По истечении интервала времени, определяемого таймером, испекшийся хлеб извлекается, а на его место поступает очередная партия сырых «заготовок». Затем процесс повторяется. График, в соответствии с которым протекает технологический процесс, обычно называют циклограммой.

Для правильной работы стенда полунатурного моделирования необходимо согласование реальной циклограммы объекта-оригинала и циклограммы моделирующей среды.

Реальная циклограмма определяет моменты времени, в которые происходит обмен данными с датчиками и исполнительными устройствами. В соответствии с реальной циклограммой строится циклограмма моделирующей среды, которая специфицирует допустимые кванты времени для выполнения моделирующих программ, чтобы обеспечить своевременную передачу данных между штатным ПО и программами моделирующей среды.

Для отработки стендовой циклограммы в состав ПО моделирующей среды включена программа-диспетчер, обеспечивающая обмен данными в заданные моменты времени. Программа-диспетчер инициирует моделирующие программы и контролирует соблюдение дисциплины синхронизации и обмена данными.

Циклограмма моделирующей среды должна быть представлена таким образом, чтобы инженер-экспериментатор мог специфицировать и/или модифицировать ее, не прибегая к ручному программированию (например, в формате текстового файла, диалоговом окне и т.п.). На основе сформированной спецификации генерируется массив данных, который программа-диспетчер использует как входной для отработки циклограммы на моделирующей ЭВМ.

Вопросы включения в стендовую циклограмму алгоритмов имитации помех, сбоев и отказов рассматриваются в последующих разделах.

4.3. Порядок проведения эксперимента

Готовый комплект моделей и сформированная циклограмма моделирующей среды позволяют приступить к следующему этапу – сборке загрузочных файлов для всех используемых на стенде ЭВМ с учетом условий планируемых экспериментов.

Для подготовки к запуску эксперимента

необходимо провести конфигурирование и настройку оборудования, а также выполнить сборку и загрузку исполняемых модулей на всех задействованных ЭВМ, после чего стенд будет готов к началу моделирования.

Непосредственно запуск подготовленного эксперимента выполняет оператор стенда, в роли которого может выступать сотрудник с ограниченным набором компетенций, ориентированных на указанную деятельность. Необходима также возможность автоматизированного запуска серии экспериментов без промежуточного вмешательства оператора, например, в ночное время.

Оператор стенда контролирует работу стенда и управляет производимыми экспериментами с помощью системы типа SCADA. Результаты моделирования накапливаются в хранилище. Детальный анализ накопленных данных выполняется в оффлайн-режиме с помощью подсистемы анализа данных моделирования.

Более детально функционирование ПП ПСиЦД с точки зрения используемого оборудования и программного обеспечения рассматривается в разделах, описывающих компоненты платформы.

4.4. Среда разработки моделирующих программ

Среда разработки предназначена для создания программных моделей, замещающих реальное оборудование на полунатурном стенде, а также для конфигурирования и сборки исполняемых модулей, обеспечивающих функционирование ПП ПСиЦД.

В случае необходимости функциональные возможности данной среды могут быть расширены за счет включения средств модификации штатного ПО. Целесообразность такого расширения очевидна при решении задачи прототипирования – когда разработчик штатного программного обеспечения по результатам моделирования видит недочеты и оперативно вносит изменения в штатные программы.

Ядром среды разработки должен быть репозиторий, в котором содержатся (и накапливаются по мере использования ПП ПСиЦД) модели разнообразных устройств и систем, библиотеки функций, реализующих протоколы обмена, и т.п. Среда разработки должна обеспечивать навигацию по репозиторию, поддержку версий и базовый набор моделей (как минимум, генераторов псевдослучайных чисел с заданным распределением).

Интерфейс среды разработки в формате панелей меню и диалоговых окон с включением графических изображений (по необходимости) должен предоставлять пользователю функциональные возможности, соответствующие современным стандартам:

– ввод и редактирование текста программы;
– компилирование и отладка программы;
– построение и редактирование циклограммы моделирования для дальнейшего использования программой-диспетчером;
– конфигурирование и сборка исполняемых модулей с использованием библиотек интерфейсных функций из состава ПП ПСиЦД и функций, реализующих дисциплину обмена данными.

Для разработки моделирующих программ необходимо наличие компиляторов для используемых языков, включая компилятор языков МЭК[8] для программ, исполняемых на ПЛК.

Завершив разработку всех требуемых моделирующих программ, необходимо определить порядок их запуска, то есть, сформировать циклограмму моделирующей среды. Данная циклограмма будет специфицировать строго упорядоченный по времени перечень действий, выполняемых программой-диспетчером в одном цикле своей работы.

Циклограмма состоит из последовательности тактов, длительность которых определяется длительностью такта в циклограмме функционирования моделируемой АСУ ТП. Для каждого такта устанавливается свой порядок вызовов процедур и обменов данными. При построении циклограммы моделирования необходимо учитывать требуемую периодичность вызовов отдельных процедур и операций приема/передачи данных при взаимодействии моделирующей среды и реального оборудования. Например, если передача параметра A происходит каждый такт, параметра B – в четных тактах, параметра C – каждый пятый такт, то циклограмма моделирования должна включать 10 тактов (рис. 4).

Параметры/ такты	1	2	3	4	5	6	7	8	9	10
A	+	+	+	+	+	+	+	+	+	+
B		+		+		+		+		+
C					+					+

Рис. 4. Пример циклограммы для параметров, передаваемых с разной периодичностью

Таким образом, число тактов, включаемых в циклограмму моделирования, является наименьшим общим кратным наименьших номеров тактов (начиная с 1), в которых выполняется вызов процедур и передача данных.

Для обеспечения возможности адаптации и повторного использования уже готовых моделей необходимо предусмотреть инструментальные средства (шаблоны программ, библиотеки функций и пр.), обеспечивающие соблюдение дисциплины синхронизации и обмена данными на этапе выполнения.

Подготовив комплект моделей и циклограмму моделирования, можно приступать к конфигурированию и сборке исполняемых модулей для ЭВМ стенда.

Интерфейс среды разработки должен учитывать потребности как программиста-разработчика моделей, так и инженера-экспериментатора, в функции которого входит конфигурирование, настройка и сборка исполняемых модулей, но не входит ручное программирование: перечисленные манипуляции инженеру-экспериментатору следует проводить с помощью скриптов и комплектов конфигурационных параметров в диалоговых окнах.

Законченным результатом использования среды разработки является комплект исполняемых модулей для всех ЭВМ, входящих в состав полунатурного стенда или цифрового двойника.

4.5. Среда выполнения эксперимента

Среда выполнения обеспечивает проведение эксперимента, в ходе которого используемое штатное ПО функционирует в связке с моделью объекта управления.

В задачи среды выполнения входит:

- загрузка и запуск исполняемых модулей на ЭВМ, входящих в состав стенда;
- отработка циклограммы моделирующей среды;
- реализация дисциплины обмена данными (синхронизация задействованных компьютеров, обеспечение и контроль передачи данных);
- отработка команд подсистемы управления выполнением, вводимых оператором процесса моделирования или заготовленных в виде программы эксперимента;
- формирование трассы (в случае режима отладки);
- передача данных о работе стенда в хранилище.

Со средой выполнения эксперимента работает оператор стенда, в роли которого может выступать инженер-экспериментатор или рядовой оператор. Интерфейс для взаимодействия со средой разработки реализуется в виде подсистемы управления выполнением.

При разработке стендов полунатурного моделирования, работающих в режиме реального времени, важно обеспечить синхронизацию часов ЭВМ, используемых в составе стенда, на протяжении всего эксперимента по моделированию, а также контролировать своевременную передачу данных между программами штатного ПО и программами моделирующей среды, поскольку при несоблюдении этих условий ухудшается качество результатов моделирования. Для выполнения данных требований применяется дисциплина синхронизации и обмена данными.

В части синхронизации часов ЭВМ в распределенной системе вычислений предусмотрена передача синхронизирующего сигнала по локальной сети ЭВМ (рис. 5). На этом рисунке двойной сплошной черной линией обозначена локальная сеть Ethernet, используемая для синхронизации часов на всех ЭВМ стенда [9].

Периодичность подачи синхронизирующего сигнала зависит от временных параметров штатной АСУ ТП. Источником этого сигнала может быть алгоритм, исполняемый на моделирующей ЭВМ.

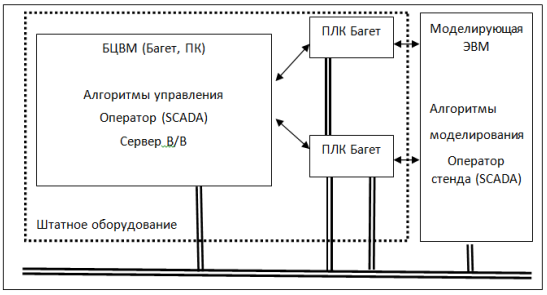

Рис. 5. Схема соединений для синхронизации часов на полунатурном стенде

В случае наличия средств синхронизации часов в составе штатной АСУ ТП допускается расширение области их действия на ЭВМ, входящие в состав стенда. Это позволит обрести требуемые функциональные возможности и одновременно сократить объем вмешательства в работу штатного аппаратного и программного обеспечения со стороны стенда.

В общем случае полунатурный стенд является распределенной системой, включающей несколько разнородных по функциональным возможностям ЭВМ, поэтому реализация интерактивного режима отладки с возможностью одновременной приостановки и пошагового выполнения программ для всех ЭВМ на стенде представляется неоправданно сложной и поэтому не включена в задачи среды выполнения. В качестве средства отладки ПО моделирующей среды предлагается использовать трассировку, для чего специфицировать отслеживаемые значения на этапе компиляции программы. Другими словами, интерактивная отладка моделирующих программ должна выполняться на этапе «черновых» версий в иных предназначенных для этого средах.

4.6. Подсистема управления выполнением эксперимента

Подсистема управления выполнением эксперимента принадлежит к классу SCADA-систем и

должна предоставлять соответствующие функциональные возможности применительно к задаче моделирования технологического процесса – планирование эксперимента, проведение эксперимента (в автоматическом режиме и «ручном» режиме с участием оператора стенда) и контроль выполнения эксперимента.

Диалоговые окна подсистемы управления выполнением эксперимента обеспечивают оператору стенда доступ к следующим возможностям:

- формирование плана эксперимента (выбор подготовленных в среде разработки исполняемых модулей, спецификация перечня запланированных отказов оборудования, помех и искажений информации, передаваемой между ПЛК и моделирующими программами, с привязкой к реальному времени, настройка средств отображения информации во время эксперимента);
- настройка аппаратной конфигурации стенда и подготовка стенда к проведению эксперимента;
- ввод команд оператора для управления работой стенда, в том числе, для загрузки и запуска исполняемых модулей, подготовленных в среде разработки;
- отображение информации о текущем состоянии стенда.

Подготовка к эксперименту начинается с составления его плана, который должен включать описание конфигурации оборудования, состав исполняемых модулей, спецификацию перечня запланированных отказов с привязкой к реальному времени и настройку средств отображения информации во время эксперимента.

Задача формирования плана эксперимента возлагается на инженера-экспериментатора.

Оператор стенда перед началом моделирования может выбрать план эксперимента и выполнить конфигурирование аппаратуры стенда, а также загрузку и начальную настройку системного и прикладного программного обеспечения.

Перечень требуемых от оператора действий и формат команд в значительной степени зависят от фактического состава стендового оборудования. Поэтому платформа ПП ПСиЦД для этих целей может предоставить только самые общие средства, например, окно терминала для выполнения скриптов с указанием параметров. Для того чтобы повысить удобство использования подсистемы управления выполнением эксперимента, следует предусмотреть возможность модификации и расширения элементов графического интерфейса подсистемы при адаптации к конкретному стенду.

На подготовленном к работе стенде запускается эксперимент и выполняется слежение за его работой в соответствии с инструкцией.

Имитация отказов и других нарушений в работе оборудования может быть выполнена программным или программно-аппаратным способом.

Программный способ удобно использовать для моделирования неполадок в каналах связи ПЛК и моделей периферийных устройств (датчиков и исполнительных механизмов). В этом случае в циклограмму моделирующей среды включается вызов процедуры, которая в определенных тактах, например, заменяет «правильные» значения передаваемых параметров на «неправильные» в соответствии с планом эксперимента.

Программно-аппаратный способ подразумевает применение специальных дополнительных устройств (например, разрывных коробок), которые встраиваются в проводные соединения стенда и физически нарушают передачу электрического сигнала в заданный момент. Для управления подобными устройствами также необходимо включить вызов соответствующей процедуры в циклограмму моделирующей среды или вывести графический интерфейс управления отказами на экран оператора стенда.

Оператор стенда способен в «ручном» режиме вносить возмущения в работу моделей, меняя значения параметров с помощью команд в диалоговом окне. Целесообразность использования данной возможности зависит от предметной области, для которой выполняется моделирование.

Команды оператора стенда предназначены для управления работой стенда и включают возможность запуска/останова процесса моделирования, а также управление форматом данных, отображаемых на экране оператора, и накоплением данных о проводимом эксперименте.

Для автоматизированного проведения серий экспериментов следует предусмотреть возможность заготовки комплектов параметров/конфигураций/скриптов.

4.7. Подсистема хранения данных по экспериментам

Подсистема хранения данных по экспериментам должна обеспечивать типовые функциональные возможности базы данных: накапливание, хранение и выборку рядов данных.

Накапливаемые данные по экспериментам должны включать не только значения параметров различных типов, но и полную информацию об условиях проведения эксперимента, включая конфигурацию аппаратуры, оборудования и план эксперимента.

Доступ к подсистеме хранения данных по экспериментам может иметь не только оператор стенда, но и другие лица – инженер-аналитик,

программист-разработчик и пр. – через дополнительный терминал.

В качестве подсистемы хранения данных может быть использован внешний программный продукт сторонних разработчиков в случае согласования форматов хранимых данных.

4.8. Подсистема анализа данных моделирования

Подсистема анализа данных моделирования должна обеспечивать универсальные функциональные возможности для цифровой обработки накопленных данных: статистические исследования, различные виды графического представления.

Базовый вариант подсистемы анализа данных моделирования может быть доработан с учетом специфики области применения АСУ ТП, в частности, для удобного графического представления данных.

Доступ к подсистеме анализа данных моделирования может иметь не только оператор стенда, но и другие лица – инженер-аналитик, программист-разработчик и пр. – через дополнительный терминал.

Подсистемы хранения и анализа данных моделирования могут быть реализованы как единое целое в виде системы хранения и обработки данных моделирования.

5. Принципы реализации программной платформы ПП ПСиЦД

Недостатком большинства стендов полунатурного моделирования и систем разработки, применяемых при создании и эксплуатации этих стендов, является уникальность наработок и невозможность повторного использования в других проектах. Поэтому основными тенденциями в области полунатурного моделирования должны стать автоматизация процесса моделирования, расширение сферы применения методов моделирования и обеспечение совместимости и повторного использования программного обеспечения за счет применения стандартов [10].

Программная платформа ПП ПСиЦД была задумана и спроектирована как аппаратно-программный каркас, на основе которого разработчик, используя предоставленный инструментарий, сможет построить и эксплуатировать модель технологического процесса.

При выборе и проектировании инструментальных средств, предлагаемых для создания полунатурных стендов на основе ПЛК «Багет», были задействованы следующие базовые принципы:

- отсутствие привязки платформы ПП ПСиЦД в исходной конфигурации к какой-либо предметной области;
- полнота функциональных возможностей для решения задачи полунатурного моделирования, начиная с разработки программ моделей и до обработки результатов моделирования;
- соответствие современным методам создания, развития и сопровождения аппаратно-программных комплексов, включая графический интерфейс и возможности конфигурирования и настройки, а также средства расширения интерфейсов;
- обеспечение достоверности и точности результатов моделирования за счет использования специальных средств синхронизации времени на ЭВМ, входящих в состав стенда, и дисциплины обмена данными на основе стандарта HLA;
- совместимость по форматам данных с существующими внешними системами, реализующими дополнительные функциональные возможности (например, графическое представление информации, обработку big data, математические методы), для использования их в режиме off-line;
- возможность модификации в плане расширения доступного набора аппаратных интерфейсов, адаптации к новой аппаратуре и т.п.;
- обеспечение возможности повторного использования программ, разработанных ранее, за счет применения шаблонов программ и библиотек интерфейсных функций.

Востребованность систем разработки полунатурных стендов в сочетании с принципами, заложенными при выборе инструментальных средств, включенных в платформу ПП ПСиЦД, будет способствовать продолжительному существованию данного программного продукта в быстро меняющихся современных условиях нашей жизни.

6. Заключение

Полунатурное моделирование как один из этапов разработки разнообразных автоматизированных систем управления имеет богатую историю и широкую область применения. В современных условиях за счет интенсивного внедрения вычислительных средств и алгоритмов автоматизации в технологические процессы роль полунатурных стендов возрастает, поскольку именно возможность включения штатного вычислительного оборудования и программного обеспечения в состав стенда повышает точность и достоверность данных, получаемых в процессе моделирования, в результате чего деятельность разработчиков сложных систем на этапах

проектирования, прототипирования и испытаний АСУ становится более эффективной и менее затратной с экономической точки зрения.

Platform for HIL Simulation Based on the Baget PLC

Svetlana Bazaeva, Yaroslav Zotov

Abstract. The article describes the project of a software platform designed for HIL simulation and digital twins based on the Baget PLC. The approach proposed by the authors provide a full set of tool for HIL simulation and a wide application range of the platform.

Keywords: software platform, HIL simulation, automated process control system, PLC “Baget”

Литература

1. Промышленные контроллеры для «Росатома» построит создатель самобытных российских процессоров. Не «Эльбрус», не «Байкал» // Интернет-издание о высоких технологиях – CNews [Электронный ресурс] Режим доступа: https://www.cnews.ru/news/top/2023-02-28_promyshlennye_kontrollery (Дата обращения: 04.09.2023).

2. ГОСТ Р 57700.22–2020 «Компьютерные модели и моделирование. Классификация»

3. ГОСТ Р 57700.37–2021 «Компьютерные модели и моделирование. Цифровые двойники изделий. Общие положения»

4. ГОСТ 16504–81 «Система государственных испытаний продукции. Испытания и контроль качества продукции. Основные термины и определения»

5. Годунов А.Н., Солдатов В.А. Операционные системы семейства Багет (сходство, отличия и перспективы) // Программирование, 2014, № 5, с. 68-76

6. Зотов Я.А. Принципы построения стенда полунатурного моделирования для отладки АСУ ТП на основе ПЛК «Багет» // Программная инженерия, 2023, принято в печать

7. 1516-2010-IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules [Электронный ресурс] Режим доступа: https://ieeexplore.ieee.org/document/5553440 (Дата обращения: 17.10.2023).

8. ГОСТ Р МЭК 61131-3-2016 «Контроллеры программируемые. Часть 3 Языки программирования»

9. Годунов А.Н., Хоменков И.И., Щепков В.Г.. Синхронизация системных часов в многопроцессорных системах // Труды НИИСИ РАН. 2021. Т. 11. № 3. С.29-35

10. Базаева С.Е., Зотов Я.А. Инструментальные средства разработки моделей технических комплексов // Труды научно-исследовательского института системных исследований Российской академии наук. 2022. Т. 12. № 4. С.5-14

Вопросы применения концепции HLA при создании стендов полунатурного моделирования

С. Е. Базаева[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, bazaeva@niisi.msk.ru

Аннотация. В статье рассматриваются вопросы применения концепции распределенного моделирования HLA (High Level Architecture) при создании стендов полунатурного моделирования для АСУ ТП. Дан краткий обзор истории возникновения HLA, преимуществ и недостатков данного стандарта применительно к поставленной задаче, проанализированы функциональные возможности, предоставляемые стандартом HLA. Автором дана оценка целесообразности применения концепции HLA при разработке инструментальных средств для полунатурных стендов на основе ПЛК Багет.

Ключевые слова: полунатурный стенд моделирования, стандарт HLA, распределенное моделирование

1. Введение

Стенды полунатурного моделирования как инструменты разработки автоматизированных систем управления имеют большую и богатую историю развития и применения.

Полунатурное моделирование возникло как одно из направлений имитационного моделирования, основная идея которого состоит в том, что структура модели соответствует структуре компонентов (подсистем) объекта-оригинала.

Метод имитационного моделирования пришел на смену математическому моделированию (но не заменил математическое моделирование полностью). Математическое моделирование в классическом формате требует формализации объекта-оригинала, например, в виде системы уравнений. Но если мы представим себе современный производственный химический комплекс, то окажется, что математическое моделирование реакции в отдельных емкостях не вызывает трудностей, однако охватить все производство в целом, да еще в реальном масштабе времени с помощью классической математической модели невозможно.

Имитационное моделирование, позволяющее «разбить» модель объекта-оригинала на компоненты-подсистемы, допускает различные способы имитации функционирования подсистем, вплоть до использования числовых рядов, полученных в ходе эксплуатации аналогичных систем или в результате вычислений, или иными способами. Некоторые подсистемы удавалось включить в процесс имитационного моделирования в штатном натурном виде – так возникло направление полунатурного моделирования.

С включением в состав АСУ ТП цифровых вычислительных средств метод полунатурного моделирования получил новый толчок к развитию, поскольку коммуникация «бортовой» цифровой ЭВМ и моделирующей ЭВМ (или моделирующего цифрового комплекса в случае большого объема вычислений) уже не представляла особой проблемы. Наличие штатного вычислительного оборудования и программного обеспечения в составе стенда способствовало повышению точности и достоверности данных, получаемых в процессе моделирования. В результате деятельность разработчиков сложных систем на этапах проектирования, прототипирования и испытаний АСУ стала более эффективной и менее затратной.

В настоящее время развитие направления полунатурного моделирования привело к формированию концепции цифровых двойников, наличие которых для вновь создаваемых АСУ ТП уже обязательно и регламентировано отечественными стандартами [1, 2, 3].

Однако вместе с указанными преимуществами метод полунатурного моделирования принес и некоторые проблемы.

Недостатком большинства стендов полунатурного моделирования и систем разработки, применяемых при создании и эксплуатации этих стендов, является их уникальность. Это обусловлено тем, что помимо цифровых ЭВМ в АСУ объекта-оригинала входит много других электрических/электронных компонентов, каждый из которых обладает своим исторически сложившимся интерфейсом и комплектом программного обеспечения, и это существенно усложняет задачу «сборки» стенда, одновременно сужая область применения созданного продукта.

Еще одной проблемой при разработке стендов полунатурного моделирования, работающих

в режиме реального времени, является обеспечение синхронизации часов ЭВМ, используемых в составе стенда, на протяжении всего эксперимента по моделированию, и контроль своевременной передачи данных между программами, которые работают на стенде.

Проблемы, связанные с разномастной аппаратурой и программным обеспечением, частично решаются по мере развития цифровых технологий: уровень совместимости компонентов АСУ ТП между собой и с оборудованием стенда на уровне оборудования возрастает за счет унификации интерфейсов и протоколов обмена данными. Одновременно увеличиваются вычислительные мощности ЭВМ, включенных в состав АСУ и моделирующего комплекса, что позволяет задействовать языки высокого уровня и программные продукты общего назначения при разработке и эксплуатации программного обеспечения – а это как минимум облегчает портирование на различные архитектуры и повторное использование наработок.

В части проблем, связанных с программным обеспечением, предлагается использовать решения, аналогичные распределенным операционным системам, которые способны контролировать циклограмму обменов данными между программами, входящими в состав системы распределенного моделирования. Одним из таких решений является концепция распределенного моделирования HLA (High Level Architecture).

2. История создания HLA

В середине 90-х годов двадцатого века при Министерстве обороны США были начаты исследования по созданию технологии, которая стала бы определяющей при построении общей архитектуры всех разрабатываемых в США систем моделирования. Для этого в 1996 году было создано специальное подразделение DMSO (Defence Modeling&Simulation Office), ведущее работы по данной тематике и отвечающее за поддержку и распространение HLA. Начиная с этого момента, всем разработчикам средств и систем моделирования было предписано следовать стандартам HLA. В 1998 году концепция HLA была предложена в качестве стандарта НАТО.

В настоящее время работы по стандарту HLA контролирует организация SISO (Simulation Interoperability Standards Organization) в координации с IEEE и OMG (Object Management Group).

Актуальная версия HLA представлена в виде стандарта IEEE 1516-2010 [4] и включает следующие документы:

– IEEE 1516-2010 – Стандарт архитектуры высокого уровня для моделирования и имитации. Структура и правила;
– IEEE 1516.1-2010 – Стандарт архитектуры высокого уровня для моделирования и имитации. Спецификации федерального интерфейса;
– IEEE 1516.2-2010 – Стандарт архитектуры высокого уровня для моделирования и имитации. Спецификация шаблона объектной модели.

Имеются также дополнительные документы, содержащие рекомендации по применению стандарта:

– IEEE 1516.3-2003 – Рекомендуемая практика разработки HLA федерации и процесс использования;
– IEEE 1516.4-2007 – Рекомендуемая практика для проверки и аккредитации федерации, как надстройка над разработкой HLA федераций и процессом использования;
– IEEE 1730-2010 – Рекомендуемая практика моделирования и процесс использования.

Поддержка стандарта IEEE 1516-2010 является обязательным требованием для работ в области имитационного моделирования и создания тренажеров, выполняемых по заказу Министерства обороны США. Данный стандарт также объявлен предпочтительным для разработок, предназначенных для эксплуатации в НАТО.

Однако объем стандарта IEEE 1516-2010 и сложность внедрения его в практику таковы, что требуется предварительный анализ и оценка целесообразности применения для решения отдельных задач.

3. Основные понятия и структура HLA

Базовые понятия стандарта HLA – это федерация, федерат, объект, атрибут и интеракция.

Объект трактуется как модель отдельного явления реального мира. Объекты не имеют методов, а имеют только состояния, которые определяются фиксированным набором атрибутов — точных значений, которые могут изменяться с течением времени. Каждый объект в любой момент времени характеризуется своим состоянием, которое определяется набором текущих значений его атрибутов.

Федерация – это объединение компонентов имитационного моделирования, называемых федератами.

В роли федератов могут выступать различные сущности:

– имитационные модели, описывающие поведение объектов программно или представленные значениями датчиков аппаратных средств;

– тренажёры, управляемые людьми;
– реальные образцы техники;
– специализированное программное обеспечение, предназначенное, например, для визуализации, сбора или обработки информации.

Федераты могут менять и получать значения атрибутов объектов. Взаимодействие федератов осуществляется посредством обмена данными.

Интеракция определяется как мгновенное сообщение (событие), не привязанное к конкретному экземпляру объекта или федерату, происходящее на уровне федерации, то есть, определить отправителя невозможно. Интеракции, в отличие от состояний объектов, не поддерживаются в системе постоянно, а имеют мгновенную природу. Примером может служить односторонняя широковещательная передача текстового сообщения всем заинтересованным участникам федерации.

Технология HLA не накладывает ограничений на внутреннее представление федератов. Федераты могут быть написаны на любом языке программирования, но должны строго соответствовать предписанному интерфейсу. Технология HLA требует, чтобы федераты обеспечивали обмен информацией, используя структуры данных, поддерживаемые сервисами RTI (RunTime Infrastucture).

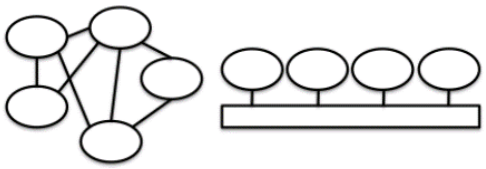

Рис. 1. Топология связей в многокомпонентной системе

Взаимодействие в многокомпонентной системе может быть организовано различными способами (рис. 1). Компоненты могут быть связаны попарно (слева на рис. 1) или с помощью общей шины (справа на рис. 1). В случае попарных связей добавление или удаление компонента системы требует модификации каналов передачи данных. При наличии общей шины манипуляции с компонентами не требуют изменения топологии существующих связей при условии соблюдения установленных правил взаимодействия через общую шину.

Рис. 2. Архитектура HLA

Архитектура HLA показана на рис. 2. Обмен данными и исполнение федератов в едином модельном времени осуществляются с помощью программной оболочки (инфраструктуры) RTI. Инфраструктура RTI является, в сущности, распределенной операционной системой для федерации. Взаимодействие федератов с RTI происходит посредством вызова федератами сервисов RTI.

FOM (Federation Object Model) – это шаблон, с помощью которого для каждого федерата декларируются общие (доступные другим федератам) данные и условия обмена данными. HLA не предписывает, какие данные должны быть включены в FOM (это ответственность пользователя федерации и разработчика), но требует соблюдения определенного формата.

Механизм взаимодействия федератов через RTI реализован в виде подписки. Федерат, заинтересованный в получении определенных атрибутов и интеракций другого федерата, должен подписаться на них через RTI.

Сервисы RTI реализуют следующие возможности:

– управление федерацией (создание федерации и управление функционированием федерации в целом);
– управление декларациями (объявление федератами экспортируемых и импортируемых данных);
– управление объектами (работа с объектами и их атрибутами;
– управление правом доступа (передача прав между федератами на модификацию значений атрибутов объектов);
– управление распределением данных (уменьшение объема данных, пересылаемых между федератами, за счет более эффективного распределения данных);
– управление временем (различные способы синхронизации локального модельного времени федератов при исполнении федерации).

4. Применение технологии HLA при разработке инструментальных средств для создания полунатурных стендов моделирования АСУ ТП

Стандарт HLA известен достаточно давно и широко применяется в системах распределенного моделирования. Преимущества использования HLA включают:

– поддержку широкого спектра программно-аппаратных платформ;
– объектно-ориентированную структуру;
– гибкость архитектуры;

– широту функциональных возможностей;
– масштабируемость.

Всё это определяет целесообразность изучения, освоения и применения архитектурных принципов HLA.

Однако внедрение данного стандарта на практике – непростая задача.

Стандарт HLA отличает большой объем информации. Приведенное описание структуры и функциональных возможностей, специфицируемых стандартом HLA, носит общий характер и не включает деталей, содержащихся в шести томах официального издания [4]. Безусловно, при изучении спецификаций HLA будут полезны дополнительные руководства [5]. Однако необходима еще и реализация RTI, которую можно выполнить самостоятельно (и это очень большая работа) или задействовать один из свободно распространяемых вариантов [6]. Имеются также коммерческие продукты [7]. Заметим, что вариант свободно распространяемого ПО не приемлем в случае работы с объектом критической инфраструктуры.

Таким образом, исследование вопроса возможности применения технологии HLA при разработке полунатурных стендов для АСУ ТП на основе ПЛК Багет [8] требует продолжения.

Базовым компонентом аппаратной части АСУ ТП на основе ПЛК Багет является программируемый логический контроллер (ПЛК), к которому подсоединены датчики и исполнительные механизмы объекта-оригинала. Программы, исполняемые на ПЛК, считывают показания датчиков, возможно, выполняют некоторую обработку этих показаний, и выдают управляющие сигналы на исполнительные механизмы. Число интерфейсов одного ПЛК может исчисляться тысячами. Количество ПЛК, задействованных в составе АСУ ТП, также может достигать нескольких тысяч. Однако наряду со сложными конфигурациями могут быть использованы и малые, состоящие из десятка ПЛК с небольшим числом интерфейсов.

Широкие возможности в плане выбора аппаратной конфигурации, спецификации дисциплины обмена данными и использования ранее разработанных моделей в роли федератов, которые обеспечивает технология HLA, являются весьма заманчивой перспективой. Однако прежде, чем приступать к внедрению HLA, необходимо выработать взвешенную оценку потенциальных накладных расходов как на стадии разработки, так и на стадии эксплуатации полунатурного стенда.

5. Заключение

Применение стандарта HLA открывает широкие перспективы при разработке инструментальных средств создания полунатурных стендов моделирования. Однако большой объем требований стандарта и сложность внедрения в процесс программирования требует более детального исследования и оценки трудоемкости планируемых работ.

Issues of Application of the HLA Concept when Developing HIL Simulation Stand

Svetlana Bazaeva

Abstract. The article describes the application of the distributed simulation concept HLA (High Level Architecture) when developing HIL simulation stands for automated process control systems. A brief overview of the history of HLA, the advantages and disadvantages of this standard in relation to the task at hand is given, and the functionality provided by the HLA standard is analyzed. The author assessed the feasibility of using the HLA concept when developing tools for HIL simulation stands based on the PLC Baget.

Keywords: HIL simulation stand, HLA standard, distributed simulation

Литература

1. ГОСТ Р 57700.22–2020 «Компьютерные модели и моделирование. Классификация».
2. ГОСТ Р 57700.37–2021 «Компьютерные модели и моделирование. Цифровые двойники изделий. Общие положения».
3. ГОСТ 16504–81 «Система государственных испытаний продукции. Испытания и контроль качества продукции. Основные термины и определения».
4. 1516-2010 – IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) –

Framework and Rules [Электронный ресурс] // Режим доступа: https://standards.ieee.org/ieee/1516/3744/ (Дата обращения: 14.11.2023).

5. Björn Möller. The HLA Tutorial. A Practical Guide for Developing Distributed Simulations [Электронный ресурс] // Режим доступа: http://pitchtechnologies.com/wp-content/uploads/2020/06/TheHLAtutorial.pdf (Дата обращения:14.11.2023).

6. CERTI Open Source HLA RTI [Электронный ресурс] // Режим доступа: https://savannah.nongnu.org/projects/certi/ (Дата обращения: 14.11.2023).

7. КПМ Ритм [Электронный ресурс] // Режим доступа: kpm-ritm.ru (Дата обращения: 14.11.2023)

8. С.Е. Базаева, Я.А. Зотов. Платформа для создания стенда полунатурного моделирования на основе ПЛК Багет. «Труды НИИСИ РАН», Т. 13 (2023), № 4

Архитектура типовой системы автоматизации технологических процессов на базе отечественных СВТ и ПО

М. С. Аристов[1], А. И. Грюнталь[2], Я. А. Зотов[3], Я. А. Шаповалов[4], Д. В. Яриков[5]

[1]НИИСИ РАН, Москва, Россия, aristov@niisi.ras.ru;

[2]НИИСИ РАН, Москва, Россия, grntl@niisi.ras.ru;

[3]НИИСИ РАН, Москва, Россия, zotov@niisi.ras.ru;

[4]НИИСИ РАН, Москва, Россия, shapovalov@niisi.ras.ru;

[5]НИИСИ РАН, Москва, Россия, yarikov@niisi.ras.ru;

Аннотация. В свете высоких рисков использования зарубежных решений для автоматизации технологических процессов критической инфраструктуры стали особенно актуальны решения на базе отечественной технологической базы. В статье описана архитектура типовой АСУ ТП на базе разработанных в ФГУ ФНЦ НИИСИ РАН программируемых логических контроллеров, операционной системы, средств разработки и другого ПО. Представленные решения поддерживают отраслевые стандарты обмена данными между АСУ ТП и устройствами управления.

Ключевые слова: программируемые логические контроллеры, операционная система, отечественное производство, автоматизированные системы управления, технологические процессы, scada, разработка ПО

1. Введение

Разработка автоматизированных систем управления технологическими процессами требует решения ряда научно-технических проблем, которые раньше решались методом заимствования и использования зарубежных компонентов СВТ и ПО.

Одними из отрицательных итогов применения зарубежных технических и технологических решений стала потеря компетенции в самостоятельной разработке отечественных обеспечивающих технологий, необходимых для реализации законченных приложений.

Использование свободно распространяемого программного обеспечения в ключевых местах элементах законченных систем привело к зависимости от технических решений, внедренных в свободно распространяемое ПО и не контролируемых РФ.

Применение свободно распространяемого ПО сдерживается отсутствием уверенности в его надежности, невозможностью гарантировать отсутствие закладных элементов, избыточной сложностью ПО, отсутствием процедур сопровождения.

Кроме того, применение зарубежного ПО сдерживается его избыточной сложностью, что, в свою очередь, является следствием универсализации программного обеспечения в соответствии с требованиями рынка.

Поэтому использование свободного программного обеспечения в ключевых элементах в системах критической информационной инфраструктуры (КИИ), а системы АСУ ТП зачастую относятся именно к этому классу систем, не представляется возможным.

Особенно жесткие требования предъявляются к операционным системам реального времени, корректное функционирование которых является основой надежности всей системы.

Соответствующие проблемы относятся и к электронной компонентной базе (ЭКБ), в первую очередь к микропроцессорам.

В связи с этим возникает задача разработки типовой структуры технологических средств для создания АСУ ТП, основанных в своих ключевых элементах на отечественном ПО и СВТ.

2. Обобщенная структура АСУ ТП

Обобщенная структура АСУ ТП приведена далее. С точки зрения функциональных элементов автоматизированная система управления технологическими процессами состоит из следующих элементов:

- программируемых логических контроллеров (ПЛК),

– автоматизированных рабочих мест оператора (АРМ оператора),
– сервера ввода-вывода.

Количество ПЛК и АРМ оператора в рамках одной системы может варьироваться и достигать десятков и сотен.

Далее описаны основные элементы, входящие в состав АСУ ТП. Более подробное описание приведено в последующих разделах. Описание структуры и назначения сервера ввода-вывода приведено в [1].

ПЛК предназначен для сбора данных, приходящих от датчиков, выработки на основе поступивших данных управляющих воздействий, направляемых к устройствам управления технологическим процессом (УУ), отправки управляющих воздействий устройствам управления (УУ). Взаимодействие ПЛК с внешними устройствами осуществляется с использованием промышленных протоколов, дискретных и аналоговых сигналов.

Кроме того, ПЛК должен обеспечивать дополнительные функции:

– протоколирование и ведение журналов системных событий,
– протоколирование и веление журналов прикладных событий,
– протоколирование и ведение журналов событий безопасности,
– самодиагностику,
– синхронизацию времени, др.

В состав ПЛК входят процессорные модули, а также модули ввода-вывода и коммуникационные модули.

Взаимодействие между процессорными модулями и модулями ввода-вывода осуществляется по скоростной шине. Один из возможных вариантов – шина EtherCat [5].

Взаимодействие между модулями ввода-вывода и внешними устройствами (датчиками и устройствами управления) происходит по дискретным или аналоговым сигналам, а также с использованием специализированных промышленных протоколов. В настоящее время, в рамках проектов ПЛК-1 и ПЛК-2 реализованы промышленные протоколы Modbus TCP, Modbus RTU, МЭК 101, МЭК 104, OPC UA.

ПЛК должен обеспечивать связь с АРМ оператора. Основное назначение АРМ оператора – оперативное отслеживание параметров технологического процесса, контроль за изменением технологических параметров, внесение изменений в технологический процесс, регистрация истории функционирования объекта управления, запись и хранение журналов событий, аварийных ситуаций.

Функционирование АРМ оператора обеспечивается программой Диспетчерского Управления и Сбора Данных (ДУиСД). Программа ДУиСД должна обеспечивать вывод параметров функционирования объекта управления, оповещение об аварийных ситуациях, интерактивный ввод и передачу на ПЛК значений параметров, управляющих состоянием и функционированием ПЛК.

Взаимодействие между программируемыми логическими контроллерами и автоматизированными рабочими местами оператора происходит с использованием сервера ввода-вывода (СВВ). Сервер ввода-вывода представляет собой промежуточное звено сбора, хранения и распространения данных, получаемых от ПЛК направляемых в АРМ оператора. Взаимодействие между ПЛК и сервером ввода-вывода (СВВ) и между АРМ оператора и сервером ввода-вывода осуществляется по проприетарному протоколу MLCP, разработанному в ФГУ ФНЦ НИИСИ РАН [11].

Ниже приведены подробные описание назначения и программной архитектуры основных элементов АСУ ТП: ПЛК, ДУиСД, сервера ввода-вывода. Описание и назначение сервера ввода-вывода приведено в [1]. Сервер ввода-вывода базируется на оригинальных кодах, без зарубежных заимствований.

3. Состав и назначение программы диспетчерского управления и сбора данных

Программа диспетчерского управления и сбора данных (программа ДУиСД) принадлежит к семейству SCADA-программ. Программа предназначена для автоматизации работы оператора технологического процесса и обеспечивает визуализацию состояния технологического процесса в виде мнемосхем, графиков, окон со значениями параметров технологического процесса объекта управления, а также передачу команд оператора одному или нескольким программируемым логическим контроллерам, которые непосредственно управляют технологическим процессом. Информационное взаимодействие между программой ДУиСД и программируемыми логическими контроллерами осуществляется по разработанному в ФГУ ФНЦ НИИСИ РАН проприетарному протоколу MLCP. Программа ДУиСД функционирует под управлением операционной системы Linux на компьютерах с архитектурой x86 и MIPS.

Программа ДУиСД включает набор технологических средств для создания мнемосхем, диалога с пользователем, автоматического накопления и хранения данных о технологическом процессе. В программе ДУиСД реализована работа

с проектами. Проекты хранятся в виде текстовых файлов в JSON-формате [12]. Программа ДУиСД функционирует в штатном и технологическом режимах.

В штатном режиме программа ДУиСД обеспечивает сбор, визуализацию и хранение значений поступающих от объекта управления параметров технологического процесса, передачу в режиме реального времени в программируемый логический контроллер команд оператора по управлению технологическим процессом, определение недопустимых отклонений значений параметров технологического процесса от штатных значений, оповещение о нештатных режимах работы объектов управления, протоколирование поступающих данных и действий оператора.

Технологический режим работы предназначен для разработки и редактирования мнемосхем объектов управления в графическом интерфейсе пользователя. В технологическом режиме обеспечивается создание и редактирование иерархической структуры объектов управления, размещение индикаторов (графиков, табло, стрелочных индикаторов и т. п.) и управляющих примитивов (кнопок, переключателей, радиокнопок и т.п.), а также настройка получаемых и отправляемых параметров технологического процесса.

Программа ДУиСД разработана в ФГУ НЦ НИИСИ РАН, основана на оригинальном программном коде, без зарубежных заимствований, и использует в своей работе штатные средства, входящие в дистрибутив ОС Linux.

4. Среда разработки и отладки прикладного ПО

Разработка прикладного программного обеспечения осуществляется с использованием специализированных языков программирования, описанных в стандарте ГОСТ Р МЭК 61131-3-2016 [13] (далее, языки МЭК). Стандарт описывает следующие языки программирования:

- LD (Ladder Diagram): применяется для реализации простых логических операций и контроля.
- FBD (Function Block Diagram): представляет собой графический язык блок-схем, где каждый блок выполняет определенную функцию.
- SFC (Sequential Function Chart): представляет собой графический язык диаграмм, которые используются для моделирования последовательных процессов, предоставляя четкую визуализацию различных этапов выполнения.
- ST (Structured Text) и IL (Instruction List): текстовые языки программирования больше похожи на традиционные языки программирования такие как Pascal и Си.

Основным достоинством этих языков является их относительная простота, практическая направленность на решение типовых задач в части АСУ ТП, стандартизованность. Несмотря на универсальность этих языков программирования, простые управляющие программы, могут написаны на них лицами, не имеющими специального программистского образования.

Основным средством программирования являются языки ST и FBD. Для разработки и отладки программ на языках программирования МЭК используется технологическая программа Среда Разработки и Отладки (СРиО). Программа СРиО исполняется в среде операционной системы Linux и позволяет создавать, отлаживать, модифицировать проекты, написанные на языках МЭК. Кроме того, среда СРиО обеспечивает применение представленных промышленных протоколов.

Среда СРиО исполняется на ЭВМ с микропроцессорной архитектурой Intel, функционирующей под управлением инструментальной операционной системы Linux.

5. Особенности реализации

Прикладное программное обеспечение (ППО) разрабатывается с использованием кросс-технологии. Процесс разработки ведется на инструментальной ЭВМ. Полученный в результате разработки исполняемый файл, называемый исполняемый образ, записывается на ПЛК, где затем и исполняется. В штатном режиме эксплуатации исполняемый образ записывается в энергонезависимую память ПЛК, откуда при старте он переписывается в ОЗУ и исполняется. Инструментальная ЭВМ при штатной эксплуатации не требуется.

Исполняемый образ содержит модули операционной системы реального времени, необходимые для функционирования ППО. В качестве операционной системы используется ОС РВ Багет 2.7, разработанная в ФГУ ФНЦ НИИСИ РАН. Также в исполняемый образ включаются средства поддержки промышленных протоколов, средства журналирования, средства разграничения доступа, другие программные компоненты, выполняемые для эксплуатации ППО.

6. Заключение

В ФГУ ФНЦ НИИСИ РАН разработан полнофункциональный программно-аппаратный комплект, предназначенный для разработки и эксплуатации автоматизированных систем управления технологическими процессами ши-

рокой номенклатуры. Комплект позволяет разрабатывать АСУ ТП без использования (заимствования) программного обеспечения других разработчиков. Комплект является типовым в том смысле, что с его помощью возможна разработка различных систем автоматизации, относящихся к разным отраслям промышленности и имеющим различное прикладное назначение.

Все ключевые элементы комплекта разработаны в РФ. Производство оборудования также находится в РФ.

Планируется работа по расширению функциональных возможностей проекта.

Architecture of a General Automated Process Control System Based on Locally Produced Computer Solutions

M. Aristov, A. Griuntal, Y. Shapovalov, D. Yarikov, Y. Zotov

Abstract. Due to high risks of using foreign-made industry automation solutions in critical infrastructure, importance of locally produced solutions can't be overestimated. The article describes a generic architecture of an automated process control system based on programmable logic controller, operating system, developer tools and other software developed by NIISI. Proposed solutions are based on international industry standards.

Keywords: programmable logic controllers, operating system, automated control system, technological process, scada, software development

Литература

1. А.Н. Онин. Программа Сервер ввода-вывода. Труды НИИСИ РАН, 2023, Т. 13, № 3, 5–12

2. Багет-ПЛК1. Доступна по адресу: https://niisi.ru/ПЛК-БАГЕТ-ПЛК1.pdf

3. Багет-ПЛК2. Доступна по адресу: https://niisi.ru/ПЛК-БАГЕТ-ПЛК2.pdf

4. Операционная система реального времени Багет 2.7: Государственная регистрация программы для ЭВМ №2023612040; заявитель и правообладатель: ФГУ ФНЦ НИИСИ РАН; заявл. 30.12.2022, опубл. 27.01.2023.

5. Industrial communication networks - Fieldbus specifications - Part 2: Physical layer specification and service definition. Доступно по адресу: https://webstore.iec.ch/publication/66933

6. Modbus Messaging on TCP/IP Implementation Guide. Доступно по адресу: https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

7. Modbus over Serial Line Specification & Implementation guide. Доступно по адресу: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

8. ГОСТ Р МЭК 60870-5-101 «Обобщающий стандарт по основным функциям телемеханики». Доступен по адресу: https://docs.cntd.ru/document/1200044527

9. ГОСТ Р МЭК 60870-5-101 «Доступ к сети для ГОСТ Р МЭК 870-5-101 с использованием стандартных транспортных профилей». Доступен по адресу: https://docs.cntd.ru/document/1200036299

10. Open Platform Communications, Unified Architecture. Доступно по адресу: https://webstore.iec.ch/searchform&q=iec%2062541

11. А. И. Грюнталь, К. Г. Нархов. Методы удаленной отладки ПЛК в среде ТСАГ СПО. Труды НИИСИ РАН, 2020, Т.10, № 5, 120–126.

12. ECMA-404. The JSON data interchange syntax. Доступно по адресу: https://ecma-international.org/publications-and-standards/standards/ecma-404/

13. ГОСТ Р МЭК 61131-3-2016 «Контроллеры программируемые. Часть 3: Языки программирования». Доступен по адресу: https://docs.cntd.ru/document/1200135008

Развитие языка Си и обзор будущего стандарта C23

В. А. Галатенко[1], Г. Л. Левченкова[2], С. В. Самборский[3]

[1]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, galat@niisi.ras.ru;

[2]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, galka@niisi.ras.ru;

[3]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, sambor@niisi.ras.ru

Аннотация. Работа содержит краткий обзор основных этапов развития языка программирования Си с момента его создания. Рассматриваются существующие стандарты этого языка. Пристальное внимание уделяется новому стандарту языка Си – C23. Выделяются его достоинства и недостатки.

Ключевые слова: язык Си, стандарт, C23

1. Введение

Язык Си был создан в 1969-1973 годах [1] – ровно полвека назад. За первые 20 лет своего существования Си стал основным языком системного программирования. Язык C++, созданный в середине 80-х годов 20-го века на основе Си, смог потеснить Си во многих сферах, но остаются области, в которых конкурировать с языком Си очень сложно.

Язык Си незаменим при написании ядер операционных систем и драйверов самых разных устройств. На этом языке реализованы сетевые протоколы, ставшие фундаментом современных компьютерных сетей. На нем запрограммированы всевозможные микроконтроллеры, которые окружают нас повсюду: в микроволновых печах и автомобилях, пылесосах и стиральных машинах.

Компиляторы языка Си существуют для всевозможных процессорных архитектур и практически для всех операционных систем. Таким образом, сложно представить ситуацию, когда прикладному программисту доступен, например, язык Java или C++, но недоступен Си.

Обратное возможно, и это необходимо учитывать, выбирая язык, если предполагается в будущем переносить программу на новые платформы.

То, что язык Си продолжает широко использоваться, заставляет этот язык развиваться. С другой стороны, то, что на языке Си пишутся программы, которые используются многие десятилетия, естественно приводит к значительному консерватизму в его развитии. Таким образом, каждый новый стандарт языка Си заслуживает внимательного изучения.

2. История возникновения языка Си

Как известно, язык Си был создан [1] для того, чтобы сделать операционную систему Unix переносимой. Существующие в конце 60-х годов языки программирования высокого уровня (FORTRAN, ALGOL, PL/I) плохо подходили для написания ядра операционной системы и драйверов устройств (есть исключения: например, операционная система Multics реализована на EPL - диалекте PL/I).

Утилиты операционной системы, такие как ls, cat или wc, можно было реализовывать на языке высокого уровня, но это было неудобно и непрактично. ОС Unix изначально создавалась для миникомпьютеров с довольно ограниченными ресурсами, поэтому было желательно, чтобы утилиты занимали как можно меньше оперативной памяти. Языки программирования высокого уровня не всегда позволяли программисту получить достаточно компактный исполняемый код. Кроме того, компиляторы «серьезных» языков высокого уровня сами требовали слишком много ресурсов (в первую очередь, памяти). На слабом компьютере такие компиляторы было сложно реализовать и работали они слишком медленно.

Альтернативой языкам высокого уровня были языки ассемблера, но использование для создания операционной системы ассемблера не только очень трудоемко, но и сразу лишает эту операционную систему переносимости. Получается операционная система для одной компьютерной архитектуры. Таких операционных систем было написано много, и все они перестали

использоваться вместе с компьютерами, для которых были написаны. Собственно, Unix изначально была написана на ассемблере PDP-7, и только четвертая версия была переписана на Си в 1973 году.

Проблемы с трудоемкостью и непереносимостью ассемблерного программирования частично решаются использованием ассемблера с развитым макропроцессором. Подобные средства были популярны при программировании на миникомпьютерах - макропроцессирование позволяло замаскировать специфические особенности разных моделей компьютеров и несколько снизить трудоемкость программирования. При помощи макросов можно было даже имитировать в ассемблере структурное программирование - использовать управляющие конструкции (условные, циклы разных видов, и т.п.) реализованные в виде макросов. Эти макросы получались весьма сложными, так как управляющие конструкции (например, циклы) могут быть вложенными и требуется не перепутать, где какая конструкция заканчивается.

В целом, подход, основанный на макроассемблерах, лишь частично снижал трудоемкость ассемблерного программирования, и не позволял добиться настоящей переносимости программ. Также затруднялась отладка, так как отлаживать приходилось уже препроцессированный код, а не исходный.

Таким образом, была потребность в языке низкого уровня, который должен быть близок к аппаратуре, чтобы программы на этом языке могли оперировать всеми ресурсами компьютера. Но поскольку язык предназначен для реализации переносимых программ, он не должен быть связан с какой-либо конкретной компьютерной архитектурой.

Кроме того, программировать должно быть намного легче, чем на ассемблере. В идеале предполагалось, что язык должен сочетать мощь языка ассемблера (в смысле доступа к архитектуре и полного контроля со стороны программиста к генерируемому коду) с удобством языков высокого уровня. Наконец, для этого языка должно быть возможно реализовать быстрый, однопроходный, и нетребовательный к ресурсам компилятор.

Понятно, что эти требования противоречивы, но именно из них и получился язык Си. Этапы раннего развития Си описаны Деннисом Ритчи (Dennis Ritchie) в статье [1].

3. Противоречивая природа языка Си

Изначально с языком Си было связано еще одно противоречие: с одной стороны Си – «переносимый макроассемблер», средство для реализации операционной системы, инструмент, используемый в системном программировании. При этом не было потребности в формальных определениях, так как, если программист не знаком ни с одной компьютерной архитектурой (адресуемая память, регистры, машинные инструкции), то он все равно никак не сможет программировать на Си. Если же программист хорошо знаком с устройством компьютера (любого) и умеет программировать на ассемблере, то ему довольно просто изучить язык Си и для этого не потребуется никаких формальных определений.

Подобный «прагматически-технический» подход доминировал в отношении языка Си первые 10-15 лет и сейчас популярен среди системных программистов и тех, кто пишет программы, которые должны работать на «голом железе» без использования операционной системы. То есть среди именно тех программистов, для которых Си и предназначен.

С другой стороны, Си можно рассматривать как полноценный язык программирования, а значит, у него должны быть точно определены синтаксис и семантика. Кроме того, должно быть описано, как именно устроен ввод/вывод и прочее взаимодействие с операционной системой. И должны быть стандартные библиотеки, а значит должны быть стандарты для языка и его библиотеки.

Можно также заметить, что если язык Си создан для написания переносимых программ, то только соответствие стандарту (точнее той части стандарта, где поведение точно определено) гарантирует эту переносимость. Иначе работоспособность программы на новом компьютере остается вопросом везения. Точное определение семантики языка необходимо также для верификации программ. В конце концов, стандарт языка необходим хотя бы для того, чтобы язык не распался на множество диалектов.

Эти два подхода, «прагматически-технический» и «научно-формализованный», неизбежно конфликтуют между собой. Как только стандарт языка точно описывает семантику языка, у компилятора появляется возможность глубоко проанализировать программу. После этого компилятор может ее агрессивно оптимизировать, и не всегда результат окажется тем, что ожидал программист-практик.

Подробнее эти проблемы рассматриваются в статье [2]. Автор замечает, что в процессе стандартизации Си приобрел много свойств обычных языков программирования и стал менее пригоден для системного программирования. Причем иногда это сделано ради того, чтобы оп-

тимизирующий компилятор мог улучшить оптимизацию на несколько процентов.

4. Краткая история стандартов языка Си

Можно считать, что язык Си сформировался в 1973 году: в нем появились структуры (тип struct), препроцессор (частично), название поменялось на «Си» (C), до этого он назывался «Би» (B), затем «Энби» (NB). Наконец, в 1973 году на Си была успешно переписана ОС Unix, что можно рассматривать как доказательство полноценности языка: в языке присутствуют все необходимые для программирования средства.

4.1. Керниган-Ритчи (K&R)

После 1973 года развитие языка продолжалось, исправлялись ошибки, в том числе довольно серьезные, например, обнаружились программы с неоднозначным синтаксическим разбором. Скажем, выражение «n=-1» можно было интерпретировать как «n = -1», или как «n = n-1». После того, как это обнаружилось, синтаксис был изменен: бинарные операторы =+, =-, и т.д. приняли современный вид +=, -=, *=, и пр. Но даже 15 лет спустя, некоторые компиляторы выдавали предупреждение на конструкции, вроде «n=-1», «x=+0.3» или даже «h=*p» (где p - указатель).

Кроме того, язык развивался, добавлялись новые возможности. Наконец, в 1978 году Брайан Керниган (Brian Kernighan) и один из основных разработчиков языка - Деннис Ритчи (Dennis Ritchie) опубликовали учебник «The C Programming Language» [3]. Эта книга стала de facto первым стандартом языка Си. Описанный в ней диалект языка Си кратко обозначается как K&R C или C78 (подчеркивая статус неофициального стандарта).

В K&R C были исправлены ошибки и включены новые возможности: добавлены новые типы данных «long int» и «unsigned int», добавлена стандартная библиотека ввода/вывода, усовершенствован препроцессор.

4.2. C89 или C90

После 1978 язык Си продолжал развиваться, многие важные возможности были добавлены разработчиками компиляторов: перечислимые типы, функции типа void (не возвращающие никакого результата), присваивание структур и функции, возвращающие структуры.

В начале 80-х годов американский национальный институт стандартизации (ANSI) начал разработку стандарта языка Си, и, после нескольких черновых вариантов, в 1989 году вышел стандарт ANSI X3.159-1989 «Programming Language C» [4], на который обычно ссылаются как на ANSI C или C89.

На следующий год вышел международный стандарт ISO/IEC 9899:1990, полностью совпадающий с C89. Таким образом, этот стандарт получил еще обозначение C90. В дальнейшем ANSI отказался от разработки собственных стандартов языка Си, так что все последующие стандарты Си - международные стандарты ISO.

В стандарт ANSI C были включены расширения, добавленные в популярных компиляторах, поддержка наборов символов, нужных для разных языков, прототипы функций, указатели типа void* (абстрактные указатели), усовершенствования препроцессора и многое другое. Тем не менее, главное отличие ANSI C от K&R C заключается в том, что ANSI C позволил указывать типы аргументов функции прямо в списке аргументов (как уже было сделано в C++), а не между списком аргументов и телом функции (как в K&R C).

Керниган и Ритчи в 1988 году опубликовали второе издание своего учебника [5], переработав его под стандарт ANSI C и добавив материал по стандартной библиотеке. Эта книга также стала классическим учебником, возможно лучшим учебником по языку Си, и была переведена на десятки языков. Русский перевод вышел в 1992 году [6].

В 1995 году был опубликовано расширение стандарта C90, получившее обозначение ISO/IEC 9899:1990/AMD1:1995, это расширение часто называют C95. Существенных дополнений немного, можно упомянуть добавления расширенной поддержки интернациональных наборов символов.

4.3. C99

Следующий за C89 значительный стандарт вышел в 1999 и называется ISO/IEC 9899:1999 [7], сокращенно C99. В этот стандарт вошло очень много полезных новинок, а именно:

- объявление переменных больше не нужно собирать в начале блока;
- новые целочисленные типы long long int и unsigned long long int;
- типы для комплексных чисел;
- массивы переменной длины;
- макросы с переменным числом аргументов;
- встраиваемые функции (inline);
- ключевое слово restrict;
- комментарии начинающиеся с // и до конца строки (заимствовано из C++);
- выборочная инициализация элементов массивов и полей структур.

Возможно самое важное дополнение в C99 - это поддержка стандарта на арифметику с плавающей точкой (IEEE 754-1985 также известен как IEC 60559). Стало возможным управлять тем, как происходит вычисление, без чего было очень

сложно, например, правильно реализовать интервальные вычисления.

4.4. C11

Следующий существенный стандарт был выпущен в 2011 году под названием ISO/IEC 9899:2011 [8], неформально C11. В этом стандарте включено много добавлений и усовершенствований, таких как:

- управление выравниванием данных: спецификатор _Alignas, оператор alignof, функция aligned_alloc;
- спецификатор функции _Noreturn;
- улучшенная поддержка кодировок UTF-16/UTF-32, типы char16_t и char32_t, функции преобразования, запись строковых констант в этих кодировках;
- ключевое слово _Generic для «type-generic expressions», можно рассматривать как условный оператор, но не по значению, а по типу, например:

```
#define cbrt(x) _Generic((x), long double: \
cbrtl, \
default: cbrt, \
float: cbrtf)(x)
```

- статические проверки (assert), т.е. проверки времени компиляции.

Тем не менее, намного важнее, то, что в C11 впервые описано стандартное поведение многопоточной программы, для чего потребовалось детально описать модель памяти.

Также были стандартизованы средства создания и управления потоками, механизмы синхронизации параллельных вычислений, в том числе мьютексы. Для создания собственных переменных потока потребовался новый спецификатор хранения _Thread_local. Для всего вышеперечисленного добавлен новый заголовочный файл <threads.h>.

В связи с добавлением многопоточности, также добавлены атомарные операции доступа к памяти, новый модификатор типа _Atomic и заголовочный файл <stdatomic.h>.

Тем самым было, наконец, исправлена странная ситуация, когда стандарт языка педантично описывает поведение программы, но только однопоточной. Тогда как множество ответственных приложений, например, в сфере телекоммуникации, - давно многопоточные.

Вслед за C11 в 2018 году вышел стандарт ISO/IEC 9899:2018 [9], который ожидался в 2017 году и, поэтому, получил неформальное имя C17 (хотя иногда его называют C18). В этом стандарте были исправлены мелкие ошибки C11 и не добавлены никакие новые возможности.

5. Новый стандарт C23 (C24)

Как легко заметить, значительные стандарты языка Си (C78, C89, C99, C11) выходили примерно с интервалом 10-12 лет. Таким образом мы вправе рассчитывать на выход нового стандарта в районе 2021-2023 годов. Действительно, в следующем 2024 ожидается выход нового стандарта языка C: ISO/IEC 9899:2024, уже давно получившего неформальное обозначение C23 [10]. Доступна окончательная черновая версия стандарта C23, и можно оценить, что именно предлагает новый стандарт.

Полный перечень того, что содержит C23, занимает 3 страницы по одной строке на каждое изменение. Следовательно, необходимо как-то систематизировать этот список.

Сразу можно заметить, что C23 отменяет или объявляет устаревшими многое из предыдущих версий Си. Здесь можно выделить два случая.

Во-первых, те особенности языка Си, которыми перестали пользоваться 30 лет назад:

- запрещено описание аргументов в стиле K&R (между списком параметров и телом функции);
- не поддерживаются представления целых чисел со знаком отличные от дополнительного кода;
- отменены триграфы (trigraphs), впрочем, и так редко используемые.

В-вторых, некоторые средства, реализованные в предыдущих версиях как макросы (например, true, false, bool, thread_local) стали ключевыми словами языка.

Очевидно, что удаление устаревших или реализованных по-новому средств не несет никакой пользы для Си-программиста. В то время, как некоторые неудобства, связанные с поддержкой старых проектов, вполне возможны.

Существенно изменилось поведение библиотечной функции realloc: раньше вызов realloc с новым нулевым размером приводил к освобождению памяти, такому же, как при использовании функции free. Теперь поведение в этой ситуации не определено (undefined).

В стандарте C23 легализованы многие возможности, которые давно стали de facto стандартными, например, поддерживаемый компилятором GCC [11] более 30 лет оператор typeof. Теперь можно пользоваться typeof не нарушая стандарт, а еще (что очень полезно) появился оператор typeof_unqual.

Также теперь позволено ставить метки внутри блока не только перед инструкциями, но и перед декларациями или в конце блока. Фактически, компиляторы это и раньше позволяли.

Кроме этого, в стандарт добавлен целый набор функций и макросов для поиска и подсчета количества бит в целых числах разного размера (например, stdc_count_ones) и включаемый файл stdbit.h. Многие подобные функции

уже были включены в стандарт POSIX, например, ffs (найти первый поднятый бит) - функция POSIX.1-2001. Но полного комплекта подобных функций не было.

Некоторые новые возможности представляют собой просто изменения, направленные на улучшение читаемости текста программы, например, запись константы теперь можно разбивать разделителем «'»: например, 0xAB'CD'EF или 123'456'789.

К такого рода возможностям отнести и поддержку двоичного форматного ввода/вывода (по аналогии с десятичным, восьмеричным и шестнадцатеричным). Теперь можно записывать десятичное число 19 как 0b10011.

Еще одно полезное нововведение – возможность не задавать имя аргумента функции, если его не предполагается использовать. Тем самым, можно лаконично указать компилятору, на то, что аргумент не используется, и это не ошибка.

Многие изменения или расширения заимствованы в C23 из C++, например, упомянутая выше свобода ставить метку в произвольное место в блоке. Некоторые возможности выглядят как несущественные украшения, например, директивы препроцессора #elifdef и #elifndef, другие представляют интерес, например, возможность, указав ключевое слово auto, не указывать явно тип инициализированной переменной.

Вероятно, самое интересное заимствование из C++ - добавление атрибутов в двойных квадратных скобках. Как минимум, данный стиль компактнее того, который использует служебное слово __attribute__. Кроме того, сразу были добавлены несколько полезных атрибутов из C++11, а также разрешено дублирование атрибутов (как в C++23).

Очевидно полезное нововведение - макросы ckd_add(), ckd_sub() и ckd_mul() позволяющие гарантировать правильный математический результат целочисленных арифметических операций. Эти макросы определены в новом включаемом файле stdckdint.h.

Остались некоторые новшества, которые пока сложно оценить. Среди них можно привести новый целочисленный тип (типы) _BitInt(N) и unsigned _BitInt(N). Будет ли это использоваться, станет ясно со временем. Пока поддержка этих типов отсутствует в последней версии компилятора GCC.

Интересное нововведение – директива препроцессора #embed, позволяющая включать в текст программы бинарные данные, автоматически преобразованные в текстовый вид. Это полезно, если требуется инициализировать массив данными содержащими изображение в формате jpeg или музыку в формате mp3. Конечно, это легко проделать внешними средствами: собственным конвертором бинарных данных в текстовые, утилитой make и обычной директивой #include. Однако, #embed удобнее тем, что не создается промежуточный текстовый файл, размер которого может в разы превышать размер исходного бинарного файла. Пока оценить пользу от директивы #embed сложно, так как ее поддержка еще отсутствует во всех реально используемых компиляторах. Также остаются нереализованными некоторые вошедшие в C23 атрибуты.

6. Критика стандарта C23

Появление черновика стандарта C23 вызвало много критики. Можно отметить статью [12]. В этой работе авторы отмечают большую пользу от макросов ckd_add(), ckd_sub() и ckd_mul(), от того что, наконец, легализован оператор typeof, и добавлена библиотека битовой обработки (stdc_count_ones и другие).

Однако затем авторы переходят к тем особенностям нового стандарта, с которыми категорически не согласны, и приходят к выводу, что риски перевешивают выгоду. Таким образом они предлагают продолжать использовать C99 или C11.

Основные претензии в этой статье выдвигаются к изменению поведения realloc при вызове с нулевым размером. Отмечается, что функция realloc являлась «швейцарским ножом», поскольку работала как функция malloc при вызове от нулевого указателя, и как free при вызове с нулевым размером. Таким образом, можно было организовать все управление памятью через realloc, и это действительно использовалось во многих старых программах.

С одной стороны, возмущение критиков понятно, но с другой стороны для старых программ можно использовать старые компиляторы, или новые в режиме совместимости со старыми стандартами, а использование подобного стиля в новых разработках является не очень хорошей идеей.

Другие проблемы, которые отмечаются в упомянутой работе, связаны со сравнением указателей, но сложно согласиться с выводами об опасности C23, так как приводимые конструкции сами довольно провокационны, например:

```
free(p);...;if(p==q){....
```

Конечно, функция free освобождает память, на которую указывает указатель p, и никак не должна модифицировать сам указатель p, поэтому последующее сравнение с q должно быть корректно. Но это однозначно очень плохой стиль, так как программист, который видит, что p используется, легко может забыть, что память уже освобождена, и заменить сравнение p==q на

p[0]==q[0], сделав, тем самым, грубую ошибку.

7. Заключение

Новый стандарт языка Си оставляет несколько противоречивое впечатление. С одной стороны, в C23 попало множество безусловно полезных новшеств, таких как надежные арифметические операции ckd_add() и другие из stdckdint.h. Кроме того, появилась масса удобных новшеств: auto вместо явного указания типа, возможность не указывать имя неиспользуемого аргумента, атрибуты в двойных квадратных скобках. Многие новшества имеют скорее декоративный характер, как сепаратор «'» в записи констант.

С другой стороны, новое поведение функции realloc() легко может привести к неработоспособности старых программ.

Опять же совершенно непонятно, будут ли использоваться новые типы _BitInt(N) и unsigned _BitInt(N). И, наконец, директива препроцессора #embed, которая выглядит довольно чужеродно в препроцессоре языка Си.

В целом создается впечатление, что авторы нового стандарта в своем стремлении узаконить практически все известные расширения, несколько перестарались, из-за чего сам стандарт выглядит довольно разнородно. Также можно отметить, что среди новинок стандарта C23 трудно выделить какую-то основную - какой в C11 была поддержка многопоточности и всего с ней связанного. Не видно в C23 и масштабных изменений, после которых можно утверждать, что появился новый язык программирования (как C++20).

Как ни странно, возможно, C23 похож на C89: также в стандарт собрано много всего разного, и также заимствованы самые заметные синтаксические новшества из C++. Но заметим, что C89 был исключительно успешный стандарт, ставший основой всех современных реализаций и стандартов языка Си.

Остается открытым еще один важный вопрос: связь между развитием программного обеспечения, в частности, языков программирования и развитием аппаратных средств. Влияние в одну сторону очевидно: при использовании современных процессоров очень важно, чтобы программа эффективно использовала кэш, предсказуемо обращалась в память и передавала управление, а также, чтобы среди исполняемых машинных команд было больше независимых друг от друга, ради их параллельного исполнения. Эти требования влияют на использование различных техник и приемов программирования, а также на использование языков программирования. Те возможности языков программирования, которые не удается эффективно реализовать, отмирают.

Однако в случае языка Си, который, (как отмечалось ранее) незаменим и вездесущ, можно заметить и обратное влияние. Если в настоящем или ближайшем будущем времени планируется разработать микропроцессор с новой оригинальной архитектурой, то несовместимость этой архитектуры с языком Си становится сильным аргументом против подобного проекта.

В качестве гипотетического примера можно привести архитектуры с неоднородной памятью, размещенной непосредственно в процессоре, или архитектуры, подобные транспьютерным, с очень небольшим объемом памяти на один вычислительный узел. В любом случае реализация языка Си для таких архитектур оказывается сложна и неэффективна. Поэтому то, что в C23 остался только дополнительный код в качестве представления знаковых целых чисел, ограничивает разработку экзотических компьютерных архитектур.

«Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).»

The Brief History of the C Language and an Overview of the Future C23 Standard

Vladimir Galatenko, Galina Levchenkova, Sergei Samborskii

Abstract. The article contains a brief overview of the main stages of the development of the C programming language since its creation. The existing standards of this language are considered. Close attention is paid to the new standard of the C language – C23. Its advantages and disadvantages are highlighted.

Keywords: C language, standard, C23

Литература

1. Ritchie Dennis M. The Development of the C Language // The Second ACM SIGPLAN Conference on History of Programming Languages (HOPL-II). 1993. ACM. pp. 201-208.

2. Yodaiken V. How ISO C became unusable for operating systems development // 11th Workshop on Programming Languages and Operating Systems (PLOS '21). 2021.

3. Kernighan Brian W., Ritchie Dennis M. The C Programming Language (1st ed.) // Englewood Cliffs, NJ: Prentice Hall. 1978.

4. C89 Standard. https://web.archive.org/web/20161223125339/http://flash-gordon.me.uk/ansi.c.txt

5. Kernighan Brian W., Ritchie Dennis M. The C Programming Language (2nd ed.) // Prentice Hall. 1988.

6. Керниган Б., Ритчи Д. Язык программирования Си: Пер. с англ. // Под ред. и с предисл. Вс.С.Штаркмана. - 2-е изд., перераб. - М.: Финансы и статистика, 1992. - 272 с.

7. C99 Standard (draft n1256). https://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf

8. C11 Standard (draft n1570). http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf

9. C17 Standard (draft n2176). https://web.archive.org/web/20181230041359/http:/www.open-std.org/jtc1/sc22/wg14/www/abq/c17_updated_proposed_fdis.pdf

10. WG14-N3096: Draft for ISO/IEC 9899:2023, April 1, 2023. https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3096.pdf

11. GNU Compiler collection. https://www.gnu.org/software/gcc

12. Terence Kelly, Yekai Pan. Catch-23: The New C Standard Sets the World on Fire // ACM Queue, March 29, 2023, Volume 21, issue 1. https://queue.acm.org/detail.cfm?id=3588242

Оценка сбоеустойчивости топологии СФ-блока на разных этапах оптимизации комбинационной логики логического синтеза

Е. К. Эмин[1], К. А. Петров[2], В. В. Азаров[3], А. П. Скоробогатов[4], А. А. Антонов[5]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, emin@cs.niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, petrovk@cs.niisi.ras.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, azarov_v@cs.niisi.ras.ru;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, skorobog_a@cs.niisi.ras.ru;

[5]ФГУ ФНЦ НИИСИ РАН, Москва, antonov@niisi.msk.ru.

Аннотация. Проведен анализ сбоеустойчивости полученных схем СФ-блока с их реальной топологической оценкой. Предложена оценка выходных характеристик полученной схемы при помощи сигмоидальной функции. Данная функция может использоваться для сравнения различных схем, а также для поиска оптимальной схемы заданной логической функции в эвристических алгоритмах, и алгоритмах машинного обучения.

Ключевые слова: сложно-функциональный блок, логический синтез, оптимизация, сбоеустойчивость, одиночные сбои, проектирование СБИС

1. Введение

Развитие электронной компонентной базы для космических аппаратов приводит к возрастанию требований к ее характеристикам, таким как вычислительная производительность, радиационная стойкость и сбоеустойчивость. В современных условиях повышение производительности часто осуществляется как за счет увеличения архитектурной сложности (увеличение количества регистров общего назначения, увеличение операций типа регистр-регистр и память-память, появление дополнительных векторных сопроцессоров и т. д.), так при помощи увеличения объема внутренней памяти. Это приводит к необходимости увеличения степени интеграции современных систем на кристалле и переходу на более низкие технологические нормы производства электронно-компонентной базы, что свою очередь, повышает уязвимость интегральных микросхем к воздействию ионизирующего излучения в космическом пространстве. В настоящее время уже на этапе разработки необходимо учитывать требования к сбоеустойчивости систем на кристалле, а дальнейшее снижение технологических норм производства приводит к необходимости поиска новых методов и средств для создания более устойчивых к внешним воздействиям интегральных схем [1].

2. Постановка задачи

Одним из наиболее распространенных методов повышения сбоеустойчивости является тройное модульное резервирование (TMR) – этот метод позволяет эффективно маскировать сбои в элементах памяти. Существует множество различных методов троирования, имеющих свои достоинства и недостатки [2,3]. Однако, для современных технологических процессов ни один из существующих методов аппаратного троирования не гарантирует полную стойкость к одиночным эффектам по причине возникновения множественных сбоев в нескольких элементах одновременно, что приводит к сбоям как в комбинационных путях, так и дереве распространения синхросигналов [4]. Предыдущее исследование [5] показало, что ни один из исследованных вариантов тройного модульного резервирования не является абсолютно универсальным.

Различают несколько механизмов маскирования сбоев в комбинационных схемах [6]:

- логическое маскирование – сбой будет маскирован, если он затрагивает часть схемы, которая логически не влияет на конечный результат;
- электрическое маскирование – сбой будет маскирован, если импульс, создаваемый попаданием частицы, затухает до того, как он достигнет входа триггера.
- временное маскирование – сбой

маскирован, если индуцированный импульс не достигнет входа триггера в момент окна фиксации.

Логический синтез является ключевым компонентом современных процессов автоматизации проектирования микросхем. Основную задачу логического синтеза можно сформулировать следующим образом: Дана логическая функция f: $B^n \rightarrow B$ (где B обозначает булево множество {0, 1}), Логическая функция f может быть реализована множеством эквивалентных логических схем. Необходимо найти логическую схему, реализующую заданную функцию f с минимальным числом логических элементов.

На практике полный перебор всех возможных решений невозможен. Исторически первый эвристический алгоритм минимизации был предложен Р. Брайнтом на основе диаграмм двоичного выбора (BDD) [7].

Данный алгоритм направлен на минимизацию систем булевых функций, что позволяет уменьшать количество элементов в комбинационных путях, тем самым снижая максимальную занимаемую площадь, и как следствие, повышать максимально возможное быстродействие синтезируемых логических функций. Сейчас в задаче логического синтеза применяют как эвристические алгоритмы (Espresso, MIS II и т. д.), так и алгоритмы машинного обучения [8].

Исследование влияния структуры схемы показало, что ее маскирующие свойства могут изменяться в зависимости от выбранных методов синтеза [9]. Поэтому в рамках данной работы предлагается рассматривать только механизм логического маскирования и оценить сбоеустойчивость на разных этапах алгоритма перебора всех возможных решений.

Существуют как коммерческие САПР логического синтеза (Design Compiler (Synopsys), Encounter RTL Compiler (Cadence Design Systems) и др.), так и проекты с открытым исходным кодом: Yosys. В коммерческих САПР при логическом синтезе учитываются параметры временной спецификации для анализа быстродействия схемы, однако их алгоритм их работы не находится в открытом доступе. В рамках данного исследования в качестве инструмента логического синтеза использовался Yosys.

На практике особую важность при моделировании последствий множественных сбоев имеет реальное топологическое размещение стандартных ячеек. Кроме того, при проектировании топологии микросхемы для достижения заданного быстродействия комбинационные пути могут подвергнуться изменениям (добавлены буферы, инверторы, меняться ветвление по входам и выходам), а также соблюдения правил проектирования. Таким образом, оценку их эффективности необходимо осуществлять с учетом точного положения стандартных ячеек на топологии микросхемы.

В рамках работ, проводимых в ФГУ ФНЦ НИИСИ РАН, был создан инструмент для моделирования последствий сбоев в ячейках с учетом топологии [10]. Он представляет собой набор классов SystemVerilog и позволяет моделировать последствия сбоев во время моделирования нетлиста. Цели для внесения сбоев могут задаваться при помощи выбора случайной координаты – инжектор генерирует квадратную область заданного размера и на основе топологической информации определяет какие ячейки оказались в этой области. Для всех пораженного последовательного элемента моделируется переключение его в противоположное логическое состояние. Сбой в комбинационном элементе (SET) – как временное изменение логического состояния (длительностью 100 пс) на выходе элемента.

2.1. Тестируемое устройство

В качестве объекта исследования был выбран 32-разрядный блок целочисленного умножения/деления, входящий в состав микропроцессорного ядра. Функционально он содержит логику для выполнения операции деления – самой длительной в тактах инструкции в составе АЛУ, во время которой не происходит промежуточного сохранения результатов в и регистрах или ОЗУ. Кроме того, его структура достаточно неоднородна, в составе нет макроблоков и элементов памяти. Это позволит адекватно оценить влияние оптимизации комбинационной логики на сбоеустойчивость. Дальнейшее исследование проводилось без изменения исходного RTL-кода.

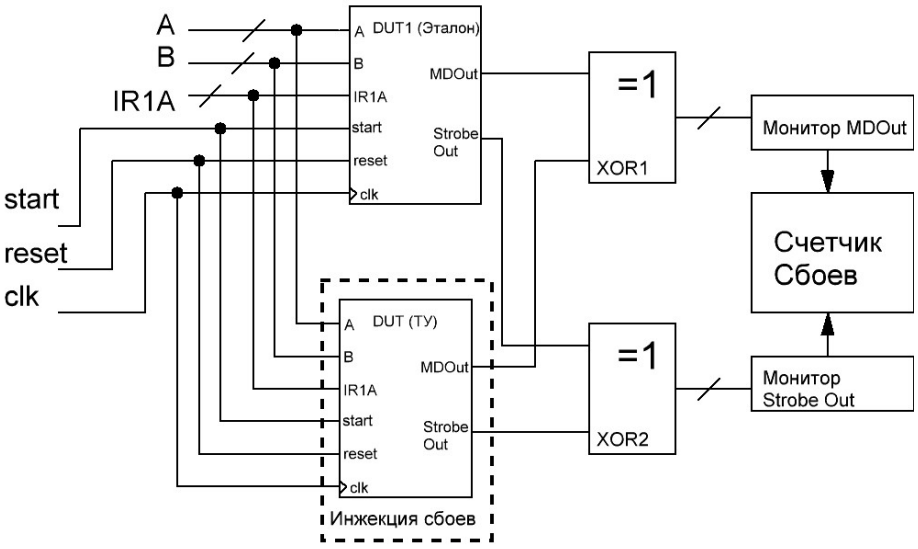

Рис. 1. Схема тестового окружения

2.2. Методика моделирования

Для тестирования каждого из вариантов была разработана тестовая схема, которая состоит из двух версий блока делителя/умножителя: эталонной и тестовой, в которую инжектировались сбои. Сначала в рамках тестовой задачи на входы A, B, IR1A (выбор соответствующего режима) обеих версий подаются случайные значения, после чего запускается тест, длительностью 35 тактов (максимальная длительность алгоритма деления). В случайный момент выполнения теста инжектируется один сбой и в пораженных ячейках моделируется одиночных эффектов. В конце теста значения выходов обеих версий сравниваются при помощи операции ИСКЛ-ИЛИ, и при любом различии выходов фиксируется ошибка. Тестирование повторяется циклически 10^4 раз для набора статистических данных.

Таблица 1. Характеристики СФ-блоков деления/умножения

N	Период, нс	Площадь, мкм^2	P_f, мВт	σ, 10^{-2} мм^2	Score
0	2,4	147946,8	46,3	0,51	3,15
10	2,2	147946,8	46,6	0,80	3,21
20	2,2	148407,2	49,9	2,39	3,44
30	2,3	148038,9	49,10	1,34	3,32
40	2,3	148038,9	49,3	1,46	3,34
50	2,2	148038,9	44,9	0,44	3,13
55	2,3	147854,8	47,7	0,57	3,16

За основу был взят маршрут логического синтеза при помощи инструмента с открытым исходным кодом Yosys, т. к. из рассмотренных инструментов синтеза только он позволяет получать промежуточные итерации минимизации логических функций.

В рамках стандартного маршрута синтез комбинационных путей выполняет другой инструмент ABC, который использует алгоритм упрощения диаграмм двоичного выбора ESPRESSO. Алгоритм является итерационным и для данного исследуемого СФ-блока занимает 55 итераций.

Для упрощения задачи проектирования топологии было решено выбрать 6 этапов с интервалом 10 итераций и последнюю итерацию. Далее была получена топология для каждой из этих схем. Характеристики этих схем приведены в таблице 1.

2.3. Результаты моделирования

Для оценки сбоеустойчивости разных вариантов использовалось сечение событий σ (в мм^2). Сечение событий – это отношение числа тестов, в которых была зафиксирована ошибка к флюенсу (1):

$$\sigma = \frac{N}{\Phi}, \quad (1)$$

где N – число тестов, завершившихся с ошибкой, Φ – значение флюенса.

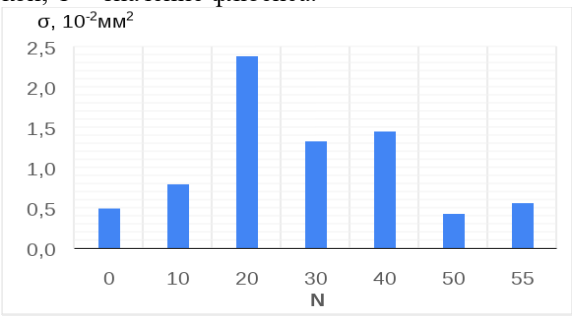

Рис. 2. Диаграмма сечения событий на разных этапах оптимизации

При проектировании топологии использовался единый изначальный коэффициент заполнения 50%. Т.к. площадь изменялась незначительно – это позволило набрать одинаковый флюенс для всех вариантов при помощи одного количества тестов $0{,}68\times10^{-5}$ мм$^{-2}$.

Сечения событий для вариантов тестируемого блока представлены на диаграмме. Как можно убедиться, с количеством итераций значение сечения событий изменяется немонотонно. Можно отметить снижение стойкости к одиночным сбоям с увеличением количества используемых функций для итераций с 1 по 20. Таким образом, можно констатировать, что увеличение количества элементов в схеме и создание подобной избыточности не всегда ведет к улучшению стойкости к одиночным сбоям, а в случае 20 итерации даже наоборот – к увеличению количества элементов, затронутых многократным сбоем и увеличению сечения событий в 4,7 раза.

Также следует отметить, что последняя итерация также не является наиболее сбоеустойчивой из рассматриваемых. А в сравнении с исходной схемой характеризуется 12% увеличением сечения событий. Самым низким значением сечения событий обладает вариант, соответствующий 50-ой итерации – $0{,}44\times10^{-2}$мм^2. Помимо этого, необходимо отметить, что он обладает наилучшим быстродействием.

Таким образом, возникает задача выбора оптимальной схемы, обладающей наилучшими характеристиками. При чем для получения оптимальной схемы необходимо учитывать начальные параметры уже на этапе логического синтеза.

В качестве решения предлагается определение численной оценки спроектированных вариантов на основе функции сигмоиды. Для формирования оценки предлагается использовать следующую функцию (2):

$$S = \alpha sigm(P) + \beta sigm(S) + \gamma sigm(P_f) + \eta sigm(\sigma), \quad (2)$$

где α, β, ɣ и η – коэффициенты влияния периода P, площади S, полной потребляемой мощности P_f и сечения событий σ. В рамках данной работы α, β, ɣ и η приняты равными 1. Данная оценка позволяет оценить “качество” полученной схемы – меньшему значению функции соответствует схема с лучшими характеристиками.

В таблице 1 для каждого из вариантов также указаны рассчитанные значения данной функции. Как можно убедиться, наилучший вариант из исследуемых – это вариант, соответствующий 50 итерации.

3. Заключение

В статье рассмотрена оценка сбоеустойчивости схем, полученных на разных этапах работы алгоритма минимизации логических функций во время операции логического синтеза. Показано, что последняя итерация соответствующая минимальной с точки зрения количества логических элементов не является наиболее сбоеустойчивой из рассматриваемых.

Самым низким значением сечения событий и наилучшим быстродействием обладает вариант, соответствующий 50-ой итерации.

Предложена оценка выходных характеристик полученной схемы при помощи сигмоидальной функции. Использование данной функции позволяет как сравнивать различные варианты схем, так и применять ее для поиска оптимальной схемы заданной логической функции в эвристических алгоритмах и в алгоритмах машинного обучения.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме FNEF-2022-0008.

Fault Tolerance Evaluation of IP-Block Topology at Different Stages of Combinational Logic Synthesis

E. K. Emin, K. A. Petrov, V. V. Azarov, A. P. Skorobogatov, A. A. Antonov

Abstract. The analysis of the fault stability of the obtained schemes with a real topological assessment is represented. Fault tolerance evaluation of the resulting circuit using a sigmoidal function is proposed. This function can be used to compare different circuits, and also used to find the optimal circuits of a given logical function in heuristic and machine learning algorithms.

Keywords: IP-block, logical synthesis, optimization, fault tolerance, single events, VLSI design

Литература

1. R. E. Bryant et al., Limitations and challenges of computer-aided design technology for CMOS VLSI // Proceedings of the IEEE, vol. 89, no. 3, pp. 341-365, March 2001.

2. Mahatme N.N., Jagannathan S., Loveless T.D., Massengill L.W., Bhuva B.L., Wen S-J. et al., Comparison of combinational and sequential error rates for a deep submicron process // IEEE Trans Nucl. Sci. (IEEE T NUCL SCI). 2011. Vol. 58, No.6. P. 2719-2725.

3. V. Petrovic and M. Krstic, Design Flow for Radhard TMR Flip-Flops // 2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems, Belgrade, Serbia, 2015, pp. 203-208.

4. P. E. Dodd, M. R. Shaneyfelt, J. R. Schwank, and J. A. Felix, Current and future challenges in radiation effects on cmos electronics // IEEE Transactions on Nuclear Science, vol. 57, no. 4, pp. 1747–1763, Aug. 2010.

5. Власов А.О., Клишин А.В., Желудков Н.В., Эмин Е.К., Горбунов М.С. Сравнительная характеристика методов повышения сбоеустойчивости топологии блоков целочисленного умножения/деления в проектных нормах 65нм // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Выпуск 3. С. 188-193.

6. Mukherjee, S. (2008). Architecture Design for Soft Errors.

7. R. E. Bryant, Graph-Based Algorithms for Boolean Function Manipulation // IEEE Transactions on Computers, Vol. C-35, No. 8 (August, 1986), pp. 677–691. Reprinted in M. Yoeli, Formal Verification of Hardware Design, IEEE Computer Society Press, 1990, pp. 253–267.

8. S. Rai et al., Logic Synthesis Meets Machine Learning: Trading Exactness for Generalization // 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2021, pp. 1026-1031.

9. Стемпковский А.Л., Соловьев Р.А., Тельпухов Д.В. Повышение сбоеустойчивости логических схем на основе частичного ресинтеза схемы // Информационные технологии. 2016. Т. 22. №7. С. 515-522.

10. P. Chernyakov et al., "Comparative Analysis of Layout-Aware Fault Injection on TMR-based DMA Controllers," // 2019 IEEE 31st International Conference on Microelectronics (MIEL), Nis, Serbia, 2019, pp. 289-292.

Сокращение энергопотребления СФ-блоков посредством автоматизированного подбора оптимальных параметров проектирования

Е. С. Кочева[1], Н. В. Желудков[2], Е. В. Ткаченко[3], Н. В. Желудков[2], К. А. Чумаков[5], К. А. Петров[6]

[1]ФГУ ФНЦ НИИСИ РАН,, Москва, Россия, kocheva@cs.niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nvgel@cs.niisi.ras.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, etkachenko@cs.niisi.ras.ru;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, eboris@cs.niisi.ras.ru;

[5]ФГУ ФНЦ НИИСИ РАН,, Москва, Россия, kchumak@cs.niisi.ras.ru;

[6]ФГУ ФНЦ НИИСИ РАН,, Москва, Россия, petrovk@cs.niisi.ras.ru

Аннотация. Исследована возможность применения метода автоматизированного подбора оптимальных параметров в процессе проектирования СФ-блоков для оптимизации их энергопотребления. В качестве объектов исследования использовались блок интерфейса ввода/вывода связи процессора и сопроцессора и блок целочисленного умножения-деления 64-разрядных чисел. Полученные результаты позволяют сделать вывод о целесообразности применения исследованного метода в маршруте проектирования СФ-блоков.

Ключевые слова: топологическое проектирование, СБИС, СФ-блок, Optuna, энергопотребление.

1. Введение

Развитие микропроцессорных систем направлено на решение поставленных вычислительных задач с максимальной эффективностью – минимальными затратами энергопотребления и стоимости. Энергоэффективность вычислительных систем выражается в соотношении затраченной энергии к максимальной производительности системы. Это делает энергопотребление одной из важнейших характеристик проектируемых микропроцессорных систем, в том числе на уровне отдельных СФ-блоков, из которых состоит микропроцессор.

Маршрут проектирования устройства включает в себя нескольких этапов. Определяющим этапом маршрута для уже разработанной поведенческой модели СФ-блока является физическое проектирование на основе нетлиста, полученного на этапе логического синтеза.

В начале этапа физического проектирования топологии разработчику необходимо определить набор входных параметров (НВП) для САПР. Результатом проектирования является топология СФ-блока с набором выходных характеристик, основными из которых являются: энергопотребление, частота работы устройства, занимаемая на кристалле площадь.

Существует ряд входных параметров, оказывающих непосредственное влияние на итоговые характеристики проектируемого СФ-блока. Примерами данных параметров являются: начальная плотность заполнения площади стандартными (библиотечными) ячейками, частота тактового сигнала, параметры сетки земли-питания, доступное число слоев металлизации для разводки и т. д.

Одним из примеров такого влияния является выходная частотная характеристика. Она в значительной степени определяется заданным в рамках НВП периодом тактового сигнала (входной параметр), в то время как об энергопотреблении (выходной параметр) сказать того же самого невозможно. Энергопотребление определяется совокупностью множества входных параметров в неявном виде.

Подбор оптимального НВП при проектировании топологии – процесс итеративный, что подразумевает, что разработчик многократно повторяет процедуру полного или частичного прохождения маршрута проектирования, подбирая оптимальный НВП в диапазонах, определяемых опытом разработчика. Опыт разработчика в данном случае заменяет полный перебор всего возможного диапазона НВП.

Проблематика решения данной задачи выражается в высокой ресурсоемкости, в том числе по времени, требуемом для полного перебора всех возможных вариантов НВП с учетом большой вариативности параметров.

Существует класс методов, позволяющих автоматизировать данный процесс путем замены

полного перебора НВП автоматизированным поиском. Это эвристические алгоритмы разной степени сложности, а также методы машинного обучения [1].

Использование метода автоматизированного подбора оптимальных параметров позволяет ранжировать эти параметры.

Для реализации метода автоматизированного подбора оптимальных параметров использовался фреймворк Optuna [2], представляющий собой фреймворк для автоматизированного поиска оптимального НВП для моделей машинного обучения. Также была определена функция числовой оценки характеристик спроектированной схемы, отражающая ключевые выходные характеристики спроектированной схемы.

В работе рассмотрена возможность применения данного метода в целях решения задач минимизации энергопотребления при проектировании СБИС с сохранением других основных выходных характеристик СФ-блока.

2. Метод автоматизированного подбора оптимальных параметров

Алгоритм, лежащий в основе метода автоматизированного подбора оптимальных параметров данного метода, представлен на рисунке 1.

Разработан управляющий сценарий (скрипт) запуска. Он представляет собой bash-скрипт (*.sh), позволяющий в автоматическом режиме запускать полный перебор входных параметров, а также в ручном режиме производить одиночные запуски маршрута проектирования, принимая на вход НВП, заданный извне. Данный скрипт также отвечает за формирование конфигурационного файла, содержащего имя блока и НВП для запуска САПР.

В целях автоматизации был использован унифицированный маршрут, который представляет собой набор скриптов (*.tcl) для прохождения каждого из этапов маршрута разработки топологии в САПР.

Для запуска алгоритма под управлением Optuna ранее был создан скрипт, позволяющий осуществлять запуски как последовательно (итерационно), так и параллельно.

По завершении каждой итерации прохождения маршрута проектирования посредством САПР производится выписка метрик, на основании которых рассчитывается числовая оценка выходных характеристик спроектированного блока с помощью Score-функции, представленной в формуле (1).

$$S = \alpha\, sigm(clk_n) + \beta\, sigm(pwr_n) + \gamma\, sigm(area_n) + \eta\, sigm(DRC_n), \quad (1)$$

где α, β, γ и η – коэффициенты влияния периода, потребляемой мощности, площади и числа DRC-нарушений соответственно; $sigm$ – сигмоидальная функция; clk_n, pwr_n, $area_n$ и DRC_n – нормированные значения соответствующих выходных параметров.

Рис. 1. Алгоритм автоматизированного проектирования на базе Optuna

Согласно Score-функции проводилась оценка оптимальности полученной схемы, где наименьшему значению соответствует схема с наилучшими выходными параметрами.

Дополнительно, в режиме поиска оптимального НВП есть возможность генерировать общий файл с информацией по всем запускам с возможностью экспорта в Excel (Report.log), а также файл, содержащий минимальное достигнутое значение Score-функции (Summary.log).

3. Параметры экспериментов

Для проведения анализа мощностных характеристик, а именно энергопотребления, были выбраны два блока, разрабатываемые ФГУ ФНЦ НИИСИ РАН по технологическим нормам 65 нм КМОП:

- cp2k_pio – блок интерфейса ввода/вывода связи процессора и сопроцессора (~1000 элементов);
- int_mult_div – блок целочисленного умножения-деления 64-разрядных чисел (~30000 элементов).

Выбор данных блоков обусловлен неболь-

шим размером, отсутствием внутренних макроблоков (массивов памяти и заказных блоков). Относительно небольшое количество внутренних элементов данных блоков обеспечивает малую длительность прохождения полного маршрута проектирования и возможность определения оптимального НВП в небольшие сроки.

3.1. Набор входных параметров

Был определен НВП, оказывающий влияние на ключевые характеристики проектируемого устройства (табл. 1).

Для проведения анализа мощностных характеристик в данной работе при расчете Score-функции согласно формуле (1) использовались следующие коэффициенты влияния:

$$\alpha = 1, \beta = 100, \gamma = 1 \text{ и } \eta = 1.$$

Превосходящий на два порядка коэффициент влияния мощности (β), относительно других коэффициентов, позволяет направить алгоритм оптимизации на минимизацию энергопотребления.

Таблица 1. Набор входных параметров

Параметр	**Диапазон значений**
Период тактового сигнала, нс	1-1,5; шаг 0,05
Максимальный фронт сигналов, нс	0,1-0,4; шаг 0,05
Плотность заполнения ячейками, %	40-70; шаг 5
Верхний металл для разводки	5-7; шаг 1
Ограничение на максимальную длину тактового сигнала	Да/Нет
Максимальная длина дерева тактового сигнала, нс	0,5-3; шаг 0,5
Особые правила для цепей тактового сигнала	Да/Нет
Экранирование цепей тактового сигнала	Нет/ одностороннее/ двустороннее
Типы ячеек с разным пороговым напряжением	svt, lvt, hvt
Типы ячеек дерева тактового сигнала	svt, lvt

4. Результаты экспериментов

На рисунках 2 и 3 представлены результаты поиска оптимального НВП (минимального значения Score-функции) посредством Optuna.

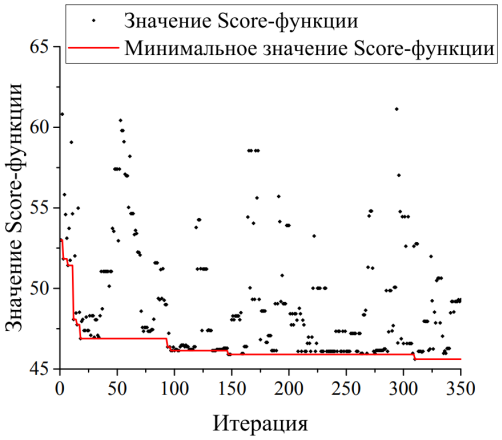

Рис. 2 – Результаты запуска фреймворка Optuna для блока cp2k_pio

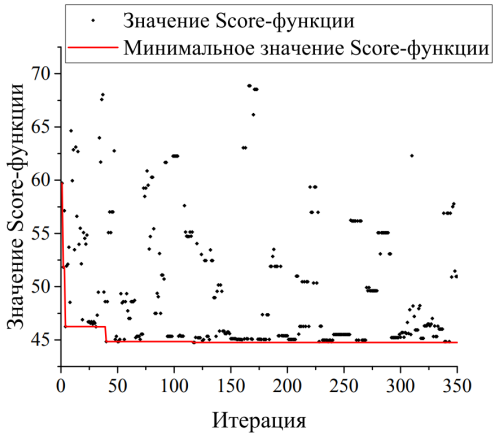

Рис. 3 – Результаты запуска фреймворка Optuna для блока int_mult_div_cpu

Точки соответствуют значению Score-функции в соответствии с номером итерации. Красная линия – индикатор минимального значения Score-функции, полученного за все предыдущие итерации.

Таблица 2. НВП при минимальном значении Score-функции для блока cp2k_pio и НВП, заданный разработчиком

Параметр	**Optuna**	**Ручное**
Значение Score-функции	45,61	45,85
Период тактового сигнала	1,45	1,50
Максимальный фронт сигналов, нс	0,3	0,3
Плотность заполнения ячейками, %	65	50
Верхний металл для разводки	7	7
Ограничение на максимальную длину тактового сигнала	Нет	Да
Максимальная длина дерева тактового сигнала, нс	–	3
Особые правила для цепей тактового сигнала	Нет	Да
Экранирование цепей тактового сигнала	Нет	Двустороннее
Типы ячеек с разным пороговым напряжением	hvt	hvt
Типы ячеек дерева тактового сигнала	svt	svt

Для блока cp2k_pio минимальное значение Score-функции (45,61) было достигнуто на 310 итерации (рис. 2) с НВП, представленным в таблице 2, наряду с НВП, заданным разработчиком.

Для блока int_mult_div_cpu минимальное значение Score-функции (44,76) было достигнуто на 117 итерации (рис. 3) с НВП, представленным в таблице 3, наряду с НВП, заданным разработчиком.

Таблица 3. НВП при минимальном значении Score-функции для блока int_mult_div_cpu и НВП, заданный разработчиком

Параметр	**Optuna**	**Ручное**
Значение Score-функции	44,76	51,25
Период тактового сигнала	1,45	1,50
Максимальный фронт сигналов, нс	0,15	0,3
Плотность заполнения ячейками, %	65	50
Верхний металл для разводки	7	7
Ограничение на максимальную длину тактового сигнала	Нет	Да
Максимальная длина дерева тактового сигнала, нс	–	3
Особые правила для цепей тактового сигнала	Нет	Да
Экранирование цепей тактового сигнала	Нет	Двустороннее
Типы ячеек с разным пороговым напряжением	lvt	hvt
Типы ячеек дерева тактового сигнала	lvt	svt

В таблице 4 представлено сравнение полученных ключевых выходных параметров для каждого из блоков при запуске с НВП, предлагаемым разработчиком и при запуске с оптимальным НВП, полученным в Optuna.

Таблица 4. Сравнение выходных параметром для блоков int_mult_div_cpu и cp2k_pio

	int_mult_div_cpu			cp2k_pio		
	Optuna	Разработчик	Разница	Optuna	Разработчик	Разница
Площадь (мм^2)	0,0088	0,2837	-23%	0,0068	0,0088	-23%
Мощность (мВт)	150,87	215,12	-30%	7,04	7,10	-1%
Частота (МГц)	680,7	374,7	+45%	704,2	671,1	+5%

4. Заключение

Исследован метод автоматизированного подбора оптимальных параметров в процессе проектирования СФ-блоков, для оптимизации их энергопотребления. Проведено сравнение результатов использования метода автоматизированного подбора оптимальных параметров с помощью фреймворка Optuna с результатами, полученных со среднестатистическим набором входных параметров. Использование метода позволило сократить энергопотребление блока cp2k_pio на 1%, а блока int_mult_div_cpu – на 30%. Незначительный выигрыш по энергопотреблению для блока cp2k_pio по сравнению с блоком int_mult_div_cpu обусловлен малыми размерами блока. При этом достигнуто сокращение площади каждого из блоков на 23%. Прирост частотных характеристик, составил 5% и 45% для блоков cp2k_pio и int_mult_div_cpu соответственно.

Полученные результаты позволяют сделать вывод о целесообразности применения исследованного метода в процессе проектирования СФ-блоков, так как полученные результаты однозначно лучше результатов ручного проектирования.

Дальнейших исследований требуют вопросы проектирования СФ-блоков большего размера, а также сравнительная оценка затрат вычислительных и трудовых ресурсов на ручное проектирование и проектирование автоматизированными методами.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме FNEF-2022-0008.

Power Consumption of VLSI IP-Blocks by Means of Automated Selection of Optimal Design Parameters

E. S. Kocheva, N. V. Zheludkov, E. V. Tkachenko, B. E. Evlampiev, K. A. Chumakov, K. A. Petrov

Abstract. The possibility of using the method of automated selection of optimal parameters in the process of designing IP-blocks to optimize their energy consumption is investigated. The I/O interface unit of the processor and coprocessor communication and the unit of integer multiplication-division of 64-bit numbers were used as objects of research. The results obtained allow us to conclude that it is expedient to use the investigated method in the design route of IP blocks.

Keywords: topological design, VLSI, IP-block, Optuna, power consumption

Литература

1. Jinwook Jung., Andrew B. Kahng. METRICS2.1 and Flow Tuning in the IEEE CEDA Robust Design Flow and OpenROAD // 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2021.

2. Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, Masanori Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework // The 25th ACM SIGKDD International Conference, 2019. С. 2623-2631.

Реализация функции пространственной фильтрации изображения на векторном сопроцессоре

С. И. Аряшев[1], П. А. Чибисов[2], В. В. Цветков[3], Д. А. Трубицын[4], К. А. Петров[5]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, aserg@cs.niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, chibisov@cs.niisi.ras.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, tsvetkov@cs.niisi.ras.ru;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, trubitsyn@cs.niisi.ras.ru;

[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, petrovk@cs.niisi.ras.ru

Аннотация. Для увеличения быстродействия универсальных микропроцессоров MIPS-подобной архитектуры в НИИСИ РАН был разработан специализированный сопроцессор, позволяющий ускорять операции с комплексными и вещественными числами одинарной и двойной точности. В статье представлены результаты применения 128-го разрядного векторного сопроцессора для задачи фильтрации изображения. На примере двух вариантов векторизации показано повышение эффективности выполнения вычислений при решении этой задачи.

Ключевые слова: векторный сопроцессор, сопроцессор вещественной арифметики, CPV, фильтрация изображений.

1. Введение

В ФГУ ФНЦ НИИСИ РАН разрабатываются универсальные и специализированные процессоры на базе архитектуры MIPS. Специализированные микропроцессоры оснащаются математическим сопроцессором цифровой обработки CPV, использование которого может дать существенный прирост производительности [1] по сравнению с вычислениями на управляющем ядре и на других вычислительных алгоритмах, связанных с массивно-параллельной обработкой больших объёмов данных.

Одной из актуальных задач для сопроцессора CPV является задача фильтрации изображений [2]. Фильтрация изображений стала важной составляющей обработки графической информации [3]. Одним из методов фильтрации изображений является пространственная фильтрация – метод фильтрации изображения, при котором обработка изображения происходит с помощью выбранного оператора последовательно к каждой точке изображения [4]. В данной работе рассматриваются два варианта реализации алгоритма пространственной фильтрации изображения.

2. Постановка задачи

Алгоритм функции пространственной фильтрации представлен в референсном си-коде, который выполняется на целочисленных исполнительных устройствах процессора. Фильтрация выполняется на монохромном изображении, представленным массивом положительных 16-ти разрядных чисел. Для каждой точки изображения вычисляется свертка окрестности этой точки размером 5x5 элементов с ядром фильтра, представляющего собой матрицу целых чисел размером 5x5 элементов. Особенностью алгоритма является то, что обрабатываемая точка находится в левом верхнем углу окрестности.

Было представлено два варианта алгоритма фильтрации изображения на векторном сопроцессоре CPV вариантах. В первом варианте, назовем его «векторизация вширь», элементы векторов собирались из соседних элементов изображения или ядра фильтра. Во втором варианте, назовем его «векторизация вглубь», изображение в горизонтальном направлении делилось на четыре фрагмента, и вектора собирались из соответствующих элементов из каждого фрагмента. Элементы векторов представлены вещественными числами одинарной точности. Размерность векторов равна четырем.

3. Первый вариант - векторизация «вширь»

В первом варианте векторизации окрестность обрабатываемой точки, включающая 25

элементов изображения, представлялась блоком из семи векторов (рис.1). Пять из них объединяют четыре верхних элемента в каждом из 5 столбцов окрестности, шестой вектор собирается из первых четырех элементов в нижней строке окрестности. Седьмой вектор включает один элемент из нижней строки окрестности. Аналогичным образом в векторном виде представляется ядро фильтра. Загрузка элементов изображения из памяти в вектора CPV выполняется в ассемблерном коде через регистры процессора с использованием команд загрузки из памяти в регистры процессора половинных слов *lhu*, команд упаковки *dins* и команд *vmtl(h)* загрузки 64 разрядного слова из регистров процессора в младшую и старшую половину регистров CPV. Преобразование данных в регистрах векторного сопроцессора в формат *float* выполняется с помощью векторной команды преобразования целых чисел в вещественные *chw*. Вычисление свертки элементов окрестности с элементами ядра фильтра выполняется с помощью команд *mvmadd*, реализующих умножение матрицы 2x2 элемента на вектор, команд суммирования элементов вектора *vsum* и команд сборки векторов *muvehl*. За один цикл загрузки элементов изображения в блок регистров CPV свертка вычисляется для 4-х элементов в строке изображения. Запись результата осуществляется с использование команд *vsdm*. В памяти результирующее изображение преобразуется в целочисленный формат.

Ниже представлено описание алгоритма в точности соответствующему референсному коду, но оптимизированного для выполнения на векторном сопроцессоре.

Из маски размером 5x5 выбирается 7 областей, как показано на рис. 1.

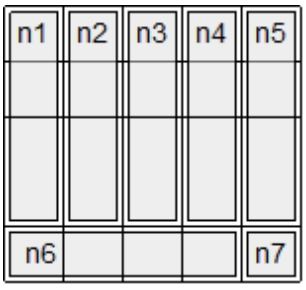

Рис. 1. 7 областей из маски 5x5

Из областей n1-n7 формируются векторы по 4 элемента (рис.2).

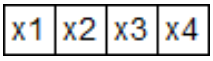

Рис. 2. Формируемый вектор

где x1, x2, x3, x4 – соответствующие значения элементов маски из покрываемой области. Для области n7 x1=x2=x3=x4 и все они соответствуют правому нижнему элементу маски. В итоге получается 7 векторов маски.

Входные данные размером WxH разбиваются на матрицы 8x5, которые состоят из четырёх подматриц 5x5 и которые разбиваются на 13 областей (рис. 3).

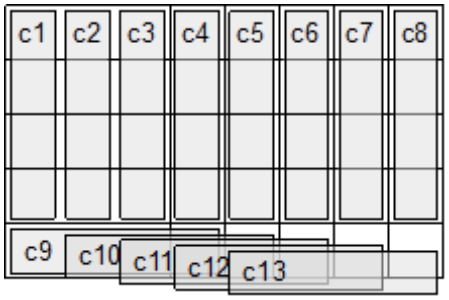

Рис. 3. 13 областей из маски 5x5

Из всех областей c1-c13 аналогичным образом формируются векторы по 4 элемента.

После формирования векторов производится расчёт одновременно для 2-х соседних результирующих точек. Расчёт проводится путём перемножения двух матриц входных данных, как показано на рис. 4, с использованием вектора маски как показано на рис. 5.

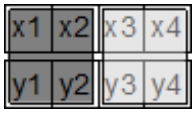

Рис. 4. Перемножение двух матриц входных данных

Рис. 5. Векторы маски для перемножения

с помощью команды векторного сопроцессора умножение матрицы на вектор с накоплением *mvmadd*: матрицы из c1 и c2 (x1/x2/y1/y2 и x3/x4/y3/y4) умножаются на векторы из n1 (z1/z2 и z3/z4). Эта операция производится для векторов из областей c1-c12 и n1-n6 с накоплением результата в два вектора R1 (первые две результирующие точки) и R2 (вторые две результирующие точки). Затем, после редукции векторов R1 и R2 в вектор R, производится финальная операция умножения с накоплением *vmadd* двух векторов c13 и вектора из области n7 с накоплением результата в вектор R. В итоге в векторе R находится результат наложения маски для 4-х соседних точек.

Код расчёта был написан на ассемблере. Изначально входные данные загружаются в стандартные регистры процессора с использованием команд *lhu* и семейства команд *dins* для полного

заполнения 64-х разрядных регистров. Затем происходит перенос данных в регистры векторного сопроцессора с помощью команд *vmtl* и *vmth*. После чего производится преобразование целых чисел в числа с плавающей точкой.

Оптимизация ассемблерного кода была направлена как на разрешение зависимостей по регистрам, так и на максимальное задействование суперскалярности. Для отслеживания условий выполнения инструкций суперскалярности и зависимостей по регистрам был написан анализатор бинарного кода. С помощью него, например, удалось добиться выполнение всего цикла расчёта без остановов конвейера процессора по причине зависимых команд.

В таблице 1 ниже представлен результат теста производительности (в тактах процессора) референсного кода и с использованием векторного сопроцессора для данных размером 1024x1024.

Таблица 1. Результаты теста производительности

Размер входных данных	1024x1024
Референсный код, такты	$116{,}5*10^6$
Векторный код, такты	$30{,}8*10^6$
Прирост, %	377

Наложение маски 5x5: 49 операций сложений и умножений => 196 операций для 4-х точек. Расчёт 4-х соседних точек на векторном сопроцессоре (одинарная точность): 16 инструкций (12 mvmadd + 3 vsum (редукция) + 1 vmadd) => 212 операций.

Для входных данных 1024x1024 пиковая производительность равна

1024 * 1024 / 4 * 16 = 4 194 304.

Таким образом при реализации первого варианта векторизации достигается производительность

4 194 304 / 30 887 459 ≈ 0,136.

В итоге, реализация приведённого алгоритма достигает 13.6% от пиковой.

4. Второй вариант - векторизация «вглубь»

Выполнение программы делится на несколько этапов (рис.6). На первом этапе изображение в горизонтальном направлении делится на 4 фрагмента и выполняется копирование исходного изображения в буфер с добавлением 4 граничных столбцов после каждого фрагмента, включающего W/4 столбцов.

Граничные столбцы дублируют первые четыре столбца в следующем фрагменте и предназначены для корректной обработки последних четырех столбцов в каждом предыдущем фрагменте. Назовем этот этап «pack1». Одновременно с добавлением граничных столбцов на этом этапе выполняется преобразование входных данных их формата *short* в формат *float*.

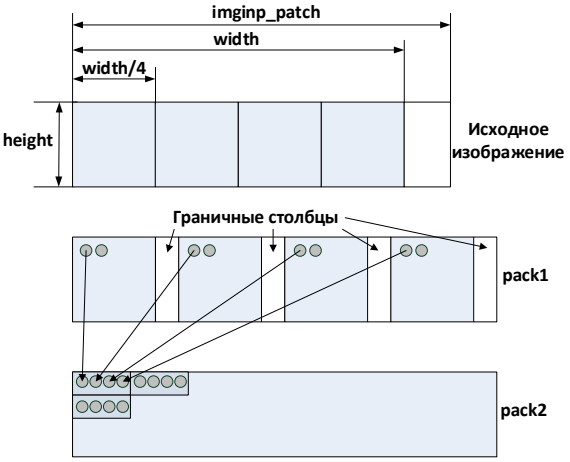

Рис. 6. Этапы pack1 и pack2

На следующем этапе выполняется упаковка этих данных. Четыре соответствующих элемента в памяти из каждого из четырех фрагментов объединяются группы по четыре элемента. Назовем этот этап «pack2». На этом этапе входные данные подготавливаются для загрузки их в регистры векторного процессора.

На следующем этапе путем выполнения векторных операций *vmadd* выполняется фильтрация одновременно во всех четырех фрагментах изображения и запись результата в формате *float* в память (рис. 7).

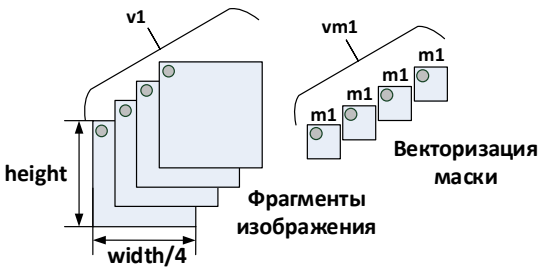

Рис. 7. Этап вычисления

На заключительном этапе – «pack3» выполняется распаковка результата и преобразование в формат *int.*

При векторизации «вглубь», как уже было сказано, изображение разбивается на четыре фрагмента и каждому фрагменту соответствует один элемент вектора. Все четыре фрагмента обрабатываются параллельно по одному и тому же

алгоритму с использованием векторных операций. Ядро фильтра представляется 25 векторами, у каждого из которых все элементы равны значению соответствующего коэффициента ядра. Элементы векторов маски могут загружаться из памяти как во время вычисления свертки, так и предварительно перед началом обработки. В первом случае под маску задействован только один регистр, оставляя доступными под загрузку элементов изображения 63 регистра. Загрузка регистров маски в реальном времени не приводит к увеличению общего времени обработки, поскольку основным фактором, определяющим время выполнения фильтрации при размерах фильтра 5x5 элементов, является время вычисления.

Цикл выполнения свертки для каждого элемента изображения разворачивается и выполняется на блоке регистров CPV. Сверка выполняется с использованием команды *vmadd* – умножение с накоплением вещественных чисел. Для каждого элемента результат накапливается в отдельном регистре. Задержка на выполнение одной команды *vmadd* составляет 5 тактов, поэтому для эффективной работы команды *vmadd* в конвейере с выдачей результата в каждом такте работы процессора необходимо выполнять одновременную обработку не менее 5 точек изображения. Одновременная обработка нескольких точек подразумевает обработку этих точек за одну загрузку блока регистров. Для вычисления свертки одного элемента изображения с фильтром 5x5 элементов требуется блок из 25 регистров. Имеющееся количество регистров позволяет выполнять одновременную обработку до 7 элементов в строке. В этом случае под загрузку элементов изображения задействуется 5x11=55 регистров CPV, семь регистров требуется для накопления результата и один регистр для маски. Общее количество задействованных регистров равно 63.

На рис. 8 представлены результаты оценки коэффициента ускорения выполнения фильтрации на этапе вычисления на векторном сопроцессоре по отношению ко времени выполнения референсного кода на целочисленных устройствах процессора при различном количестве одновременно обрабатываемых точек изображения. Фильтрация выполнялась на изображении размером 1000x1000 элемента. До значения количества обрабатываемых точек равном 4 под маску выделялось 25 регистров, и загрузка регистров маски выполнялась до начала вычислений. При дальнейшем увеличении количества обрабатываемых точек часть регистров маски загружались во время вычислений и количество выделяемых под маску регистров сокращалось вплоть до одного при количестве одновременно обрабатываемых точек равном 7. Видно, что коэффициент ускорения увеличивается с увеличением количества обрабатываемых точек. Максимальное значение достигается при 4 точках и превышает 10. При дальнейшем увеличении количества обрабатываемых точек коэффициент ускорения незначительно снижается.

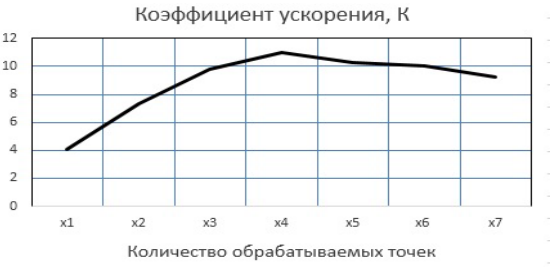

Рис. 8. Зависимость коэффициента ускорения от количества обрабатываемых точек

5. Сравнительная оценка двух вариантов реализации алгоритма фильтрации изображений

Для сравнения производительности двух алгоритмов время их выполнения сопоставлялось с временем выполнения референсного кода на целочисленных исполнительных устройствах процессора. Время выполнения отдельных этапов можно оценить исходя из коэффициента ускорения и измеренного времени выполнения референсного кода, которое при параметрах изображения width=1000, height=1000 составляет $116{,}5*10^6$ тактов.

Результаты оценки ускорения выполнения фильтрации приведены в таблице 2. Измерения коэффициентов ускорения выполнялось при моделировании работы процессора и векторного сопроцессора на ПЛИС ALTERA. Функция фильтра вызывалась со следующими параметрами:

width = 1000, height = 1000,
imginp_pitch = 1004, imgout_pith = 1000,
shift = 0, round = 0, thr = 1900000.

Таблица 2. Сравнение коэффициентов ускорения двух вариантов векторизации

Коэффициент	Вариант 1	Вариант 2
K	3.8	11
K1	-	6,1
K2	-	3,6
K3	-	2,1

Для второго варианта определены четыре коэффициента ускорения К, К1, К2 и К3. К – максимальный коэффициент ускорения только на этапе вычислений в ситуации, когда элементы исходного результирующего изображений представлены в формате *float* и упакованы в соответствии с концепцией векторизации вглубь. К1 – коэффициент ускорения вычислений с учетом времени потраченного на упаковку элементов входного изображения, которое уже представлено в формате *float* (этап «pack2»). К2 - коэффициент ускорения вычислений с учетом времени потраченного на преобразование входного изображения во *float* и упаковку (этапы pack1 и pack2**).** К3 - коэффициент ускорения с учетом всех преобразований, то есть когда входное изображение представлено в формате *short*, а результирующее в формате *int*. Такие форматы входного и результирующего изображения приняты в референсном коде и в первом варианте реализации алгоритма фильтрации на CPV. Оценка ускорения только на вычислительном этапе представляет интерес, поскольку преобразование и упаковка (этапы pack1 и pack2) могут быть выполнены на фоне загрузки изображения в память, а необходимость выполнения обратного преобразования (этапа pack3) определяется тем, каким образом используется результат фильтрации в дальнейших вычислениях.

Как видно из таблицы 2 на этапе вычисления ускорение выполнения фильтрации на CPV по отношению к выполнению референсного кода на целочисленных исполнительных устройствах процессора во втором варианте алгоритма равно 11. Время выполнения этапа вычислений во втором варианте, то есть без учета времени, потраченного на преобразования, составляет примерно 10^7 тактов и существенно меньше времени выполнения фильтрации в первом варианте реализации алгоритма. С учетом времени, потраченного на прямое и обратное преобразования массивов, коэффициент время выполнения фильтрации во втором варианте алгоритма примерно в два раза больше, чем в первом варианте. Следует отметить, что преобразования форматов и упаковка массивов реализована на уровне компиляции и дополнительных усилий по оптимизации этих процедур предпринято не было, так как данная задача уже решалась в рамках [5].

Оценим эффективность выполнения вычислений на векторном сопроцессоре во втором варианте реализации алгоритма. Эффективность организации вычислений будем оценивать, как процентное отношение реально достигнутой производительности к максимально возможному (пиковому) значению. Это отношение можно вычислить как отношение реально выполненного количества вычислительных вещественных операций за такт к теоретически возможному максимальному количеству вычислительных вещественных операций за такт, которое можно реализовать при вычислениях.

При выполнении фильтрации монохромного изображения с размером фильтра m*m для каждой точки изображения выполняется m^2 операций умножения и m(m-1) операция сложения. Для изображения размером *height* * *width* необходимое количество вычислительных операций равно:

$$N \approx (2m^2)* \textbf{ height * width},$$

что при значении параметров: height = 1000, widths = 1000, m = 5 составляет $5*10^7$.

При выполнении фильтрации на векторном сопроцессоре во втором варианте реализации алгоритма при этих значениях параметров затрачивается, как было показано выше, примерно 10^7 тактов. То есть в этом варианте на вычислительном этапе выполняется 5 вычислительных вещественных операций за такт. Максимальное теоретически возможное (пиковое) количество операций за такт равно 4x2=8. Коэффициент 4 – это размерность вектора (*float*), а коэффициент 2 – вклад использования векторной команды *vmadd*. Таким образом, на вычислительном этапе во втором варианте достигается производительность 5/8*100% = 62,5% от пикового значения.

В итоге, реализация приведённого алгоритма фильтрации достигает 62,5% от пиковой.

6. Заключение

В статье были рассмотрены два варианта алгоритмов пространственной фильтрации изображения с помощью векторного сопроцессора. Была проведена сравнительная оценка каждого из вариантов фильтрации.

Предложенные алгоритмы позволили повысить производительность выбранной задачи пространственной фильтрации изображения. Первый вариант – векторизация «вширь» – позволил достичь ускорения выполнения фильтрации в 3,8 раза, а второй вариант – векторизация «вглубь» – в 2,1 раза при решении оригинальной задачи по сравнению с решением на управляющем ядре. При этом большая часть выигрыша в производительности теряется на переконвертацию входных и выходных данных, в то время как выполнение непосредственно вычислительной задачи ускоряется в 11 раз. Корректировка исходной задачи под выполнение на векторном сопроцессоре избавляет от необходимости конвертации данных и позволяет достичь максимальной производительности при использовании CPV.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2022-0004.

Implementation of Spatial Image Filtering Function on a Vector Coprocessor

S. I. Aryashev, P. A. Chibisov, V. V. Tsvetkov, D. A. Trubitsyn, K. A. Petrov

Abstract. To increase the performance of universal microprocessors of MIPS-like architecture, NIISI RAS developed a specialized coprocessor that allows accelerating operations with complex and real numbers of single and double precision. The article presents the results of using a 128-bit vector coprocessor for the image filtering task. Using the example of two vectorization options, the increase in the efficiency of calculations when solving this problem is show

Keywords: vector coprocessor, CPV, image filtering

Литература

1. П.Б. Богданов, О.Ю. Сударева. Применение отечественных специализированных процессоров семейства КОМДИВ в научных расчетах // Информационные технологии и вычислительные системы 3/2016.

2. S. Sadangi, S. Baraha, D. K. Satpathy and P. K. Biswal, "FPGA implementation of spatial filtering techniques for 2D images," 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 2017, pp. 1213-1217, doi: 10.1109/RTEICT.2017.8256791.

3. D. H. Rao and P. P. Panduranga, "A Survey on Image Enhancement Techniques: Classical Spatial Filter, Neural Network, Cellular Neural Network, and Fuzzy Filter," 2006 IEEE International Conference on Industrial Technology, Mumbai, India, 2006, pp. 2821-2826, doi: 10.1109/ICIT.2006.372671.

4. Н.С. Сергеев. Пространственная фильтрация изображений в системах технического зрения.

5. П.Б. Богданов, О.Ю. Сударева. Декодирование изображений в формате JPEG на процессорах КОМДИВ // Информационные технологии и вычислительные системы 1/2019.

Оценка карты разводимости при проектировании цифровых блоков СБИС с помощью графовых нейронных сетей

Н. В. Желудков[1], Я. М. Карандашев[2], Е. С. Кочева[1], М. Х. Сайбодалов[1], З. Б. Сохова[1], А. А. Умнова[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nvgel@cs.niisi.ras.ru;

[2] ФГУ ФНЦ НИИСИ РАН, Москва, Россия, karandashev@niisi.ras.ru

Аннотация. В рамках данной работы рассматривается решение задачи оценки карты разводимости на ранних этапах топологического проектирования цифровых блоков СБИС с помощью применения нейросетевой модели машинного обучения, основанной на графовой нейронной сети. Раннее предсказание проблемных мест с разводкой позволит разработчику топологии изменить такие характеристики проектируемого блока, как план размещения, расположение макроблоков, а также входных и выходных портов таким образом, чтобы предотвратить возникновение проблем с трассировкой соединений на поздних этапах, тем самым сократив число запусков САПР и общее время проектирования схемы. Применение графовых нейронных сетей позволяет учитывать дополнительную информацию о связях элементов в нетлисте, для более точного предсказания.

Ключевые слова: СБИС, топологическое проектирование, карта разводимости, машинное обучение, графовые нейронные сети

1. Введение

Одним из этапов топологического проектирования цифровых блоков СБИС является оценка разводимости межсоединений элементов на кристалле. Данный этап в САПР заключается в следующем: происходит разделение площади схемы на равные участки в виде квадратов со стороной в несколько высот стандартных ячеек, оценивается число межсоединений, которые должны проходить через этот квадрат, определяется число доступных для разводки треков металлизации в этой области, вычисляется отношение первого значения ко второму. Если это соотношение не превышает 100%, то нарушение разводимости в рассматриваемой области отсутствует – доступного числа треков хватает для разводки требуемого числа трасс металлизации. При значении этого соотношения больше 100% существенно вырастает вероятность получить нарушение DRC-правил (правил проектирования для выбранной технологии) после проведения этапа детальной трассировки. В рамках данной работы мы будем называть данное соотношение как “разводимость”, а агрегацию локальных значений этого параметра по всей площади – “картой разводимости”. Аналогичным параметром в англоязычной литературе и САПР является “congestion”.

Оценка разводимости в современных САПР проводится после прохождения двух этапов топологического проектирования:

- размещения стандартных ячеек (включая глобальное и детальное размещение);
- глобальной трассировки, которая заключается в оценочной трассировке межсоединений без сильной привязки к трекам.

Таким образом, получить карту разводимости, чтобы оценить проблемные места, можно только после прохождения в САПР перечисленных выше этапов, что может занимать для современных блоков СБИС с сотнями тысяч и миллионов логических вентилей значительное время. Это создает следующую проблему: для оценки разводимости при любом изменении плана размещения и новой расстановкой макроблоков, при изменении в сетке земли-питания, а также изменении изначального нетлиста схемы – требуется запускать новую итерацию маршрута в САПР с прохождением этапов размещения стандартных ячеек и глобальной трассировкой. Запуск каждой итерации САПР для оценки карты разводимости на новом плане размещения увеличивает общее время проектирования схемы.

Основной задачей, решаемой в рамках данной работы, является создание модели машинного обучения для предсказания карты разводимости без прохождения времязатратных этапов размещения стандартных ячеек и глобальной трассировки. Входными данным для этой мо-

дели являются изначальный нетлист проектируемой схемы, а также характеристики стандартных ячеек, представленных в технологической библиотеке – число пинов и площадь ячеек. Для получения выходной метрики (значения разводимости на локальных участках) для обучения модели был создан маршрут топологического проектирования в САПР с открытым исходным кодом OpenROAD [1], включающий в себя прохождение этапов размещения ячеек, глобальной трассировки и выписки реальной карты разводимости. На Рисунке 1 представлено изображение карты разводимости, полученной в САПР OpenROAD.

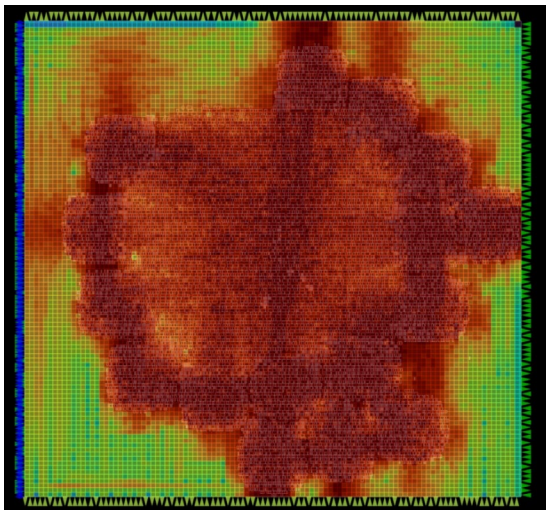

Рис. 1. Карта разводимости в OpenROAD

Более темным участкам соответствуют места на плане размещения с большим значением разводимости.

Основными причинами нарушения разводимости можно считать:

- присутствие в нетлисте логических вентилей с большим числом пинов, что может сказаться на разводимости в том месте, где будут размещены в дальнейшем эти ячейки, особенно если несколько таких ячеек будут расположены рядом друг с другом;
- высокая плотность сетки земли-питания, трассы которой занимают большое число треков, доступных для разводки;
- размещение большого числа стандартных ячеек в узких каналах между макроблоками и рядом с их краями;
- плотное размещение входных и выходных портов, что влечет за собой повышенную плотность разводки в этой области.

Для обучения и тестирования модели был выбран набор из 25 блоков, находящийся в открытом доступе (openABC) [2] и представленный в виде RTL описания. Был произведен логический синтез блоков в САПР Genus на основе технологии “Микрон 180 нм”, а также выполнено топологическое проектирование по этим же проектным нормам для получения реальных значений карт разводимости.

В своей работе мы опирались на статьи Кирби и др. [3] и Гозе и др. [4]. В этих работах авторы получали данные из нетлистов и представляли их в качестве графов, где у каждого узла есть некоторые атрибуты. В данной работе используется тот же подход: из нетлистов мы получали данные о площади ячеек и количестве пинов для каждой ячейки и использовали эти данные для обучения наших моделей. Значения разводимости, полученные из карты разводимости, использовались в качестве меток. Именно эти значения мы будем предсказывать с помощью нейронных сетей в п.3.

2. Методы

2.1. Графовые нейронные сети

Графовые нейронные сети (GNN) представляют собой класс нейронных сетей, предназначенных для обработки данных, представленных в виде графа. Основная идея GNN заключается в обновлении признаков каждой вершины графа на основе признаков ее соседей. GCN (Graph Convolutional Network) является самым распространенным видом графовых нейронных сетей. На каждом слое GCN вершины обмениваются информацией с их соседями и обновляют свои признаки. В данной работе мы применяли два типа графовых слоев: GATConv [5] и SAGEConv [6] из фреймворка PyTorch Geometric [7].

Рассмотрим граф $G = (V, E)$, где G – набор узлов, а E – набор ребер. Матрицей смежности A мы называем матрицу, в которой $A_{ij} = 1$, если существует ребро $(i, j) \in E$.

Базовый слой SAGEConv определяется следующим образом:

$$h_i^{(k)} = f^{(k)}(W_1^{(k)} h_i^{(k-1)} + W_2^{(k)} MEAN\left(h_j^{(k-1)}\right), \quad (1)$$

где $W^{(k)}$ – матрица весов, а $f^{(k)}$ – нелинейная функция активации, например, ReLU.

Слой GATConv, в свою очередь, реализует концепцию внимания (attention), аналогично тому, как этот механизм используется в других классах нейронных сетей:

$$e_{ij}^{(k)} = \text{LeakyReLU}\left(\left[W^{(k)} h_i^{(k-1)} || W^{(k)} h_j^{(k-1)}\right]^T a^{(k)}\right), \quad (2)$$

$$\alpha_{ij}^{(k)} = \mathrm{softmax}_j\left(e_{ij}^{(k)}\right) = \frac{\exp\left(e_{ij}^{(k)}\right)}{\sum_{r\in\mathcal{N}_i}\exp\left(e_{ir}^{(k)}\right)}, \quad (3)$$

$$h_v^{(k)} = f^{(k)}\left(W^{(k)}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^{(k)}\,h_j^{(k-1)}\right). \quad (4)$$

2.2. Обработка данных

Каждый элемент, полученной ранее схемы, содержащийся в нетлисте, рассматривается как узел в графе. Рёбра представляют собой взаимосвязи между элементами схемы, как определено в проводах нетлиста. Мы предполагаем, что набор узлов в каждом проводе полносвязный. Входные и выходные порты удаляются из графа, так как они обычно размещаются вручную, и их высокие степени узлов по сравнению со стандартными ячейками негативно сказываются на эффективности обучения. Таким образом, в качестве элементов схемы мы оставляем только ячейки. Провода со степенью более 10 исключаются из итогового графа, так как они вводят клики, слишком большие для эффективной работы с ними. Полученный нетлист является графом, так как он представляет из себя набор ячеек и соединения между ними. Соответственно, мы можем представить нетлист как граф $G = (V, E)$, где V – набор узлов, представляющий ячейки и E – набор ребер. Каждый узел V имеет набор атрибутов $X \in R^3$, состоящий из значений площади ячейки (area) и количества соединений (num_pins) – входные параметры, а также значения разводимости (congestion value) – значения, которые мы собираемся предсказывать с помощью нейронных сетей. Значение разводимости мы получаем из карты разводимости путем присвоения ячейкам значений из соответствующих квадратов на карте, так как каждая ячейка попадает в некоторый квадрат на карте разводимости. Таким образом, были обработаны все 25 нетлистов и получено столько же графов для дальнейшего обучения моделей. Характеристики получившихся графов можно увидеть в Приложении 1. Хотя показатель разводимости в большинстве случаев варьируется в пределах от 0 до 100, обнаружен граф, где этот показатель достигает 518. Данный граф был исключен из выборки, так как подобные значения негативно влияют на обучение модели.

3. Результаты

В качестве тестовой выборки были выбраны 5 графов. Мы сравнили результаты моделей, в которых использовали слои GAT, SAGE и простой линейный персептрон с методом аппроксимации Neighbourhood size, в котором мы вычисляем количество соседей в окрестности 5 переходов, как было предложено в [8]. Результаты, усредненные по 10 обучениям моделей, представлены в Таблице 1. В качестве метрик работы алгоритмов мы приводим значения корреляции Кендалла и средней абсолютной ошибки (MAE).

Корреляция Кендалла вычисляется путем оценки того, является ли порядок выборки пар одинаковым как в реальных, так и в прогнозируемых значениях для всех возможных пар. То есть это разница между процентом совпавших и инверсных пар. Таким образом, необязательно, чтобы распределение предсказаний соответствовало распределению реальных значений.

Средняя абсолютная ошибка – это среднее абсолютных разностей между целевым значением и значением, предсказанным моделью. Таким образом, мы видим, насколько в среднем модель ошибается в каждом предсказании. На основании трех обученных алгоритмов были построены карты предсказаний разводимости. Результаты представлены на Рисунке 2.

Таблица 1. Результаты работы моделей

	ac97_top	aes128_core	dft_top	eth_top	fpu
	Корреляция Кендалла				
GAT(3 слоя, 2 MLP)	0.137 +- 0.005	0.239 +- 0.025	0.250 +- 0.014	0.076 +- 0.013	0.202 +- 0.009
SAGE(3 слоя, 2 MLP)	0,155 +- 0.007	0,294 +- 0.007	0.280 +- 0.008	0.048 +- 0.029	0.184 +- 0.010
Linear(2,1)	0.140 +- 0.022	0.207 +- 0.571	0.144 +- 0.196	0.053 +- 0.381	0.169 +- 0.644
Neighbourhood metric	0.198	-0.052	0.482	0.232	0.194
	Средняя абсолютная ошибка				

GAT(3 слоя, 2 MLP)	13.52 +- 0.33	11.49 +- 0.57	10.86 +- 0.19	10.96 +- 0.38	10.35 +- 0.64
SAGE(3 слоя, 2 MLP)	12.73 +- 1.01	9.83 +- 0.64	10.70 +- 0.70	11.73 +- 1.01	8.10 +- 0.15
Linear(2,1)	16.26 +- 3.21	9.29 +- 2.85	12.02 +- 0.50	10.87 +- 1.18	17.83 +- 3.82

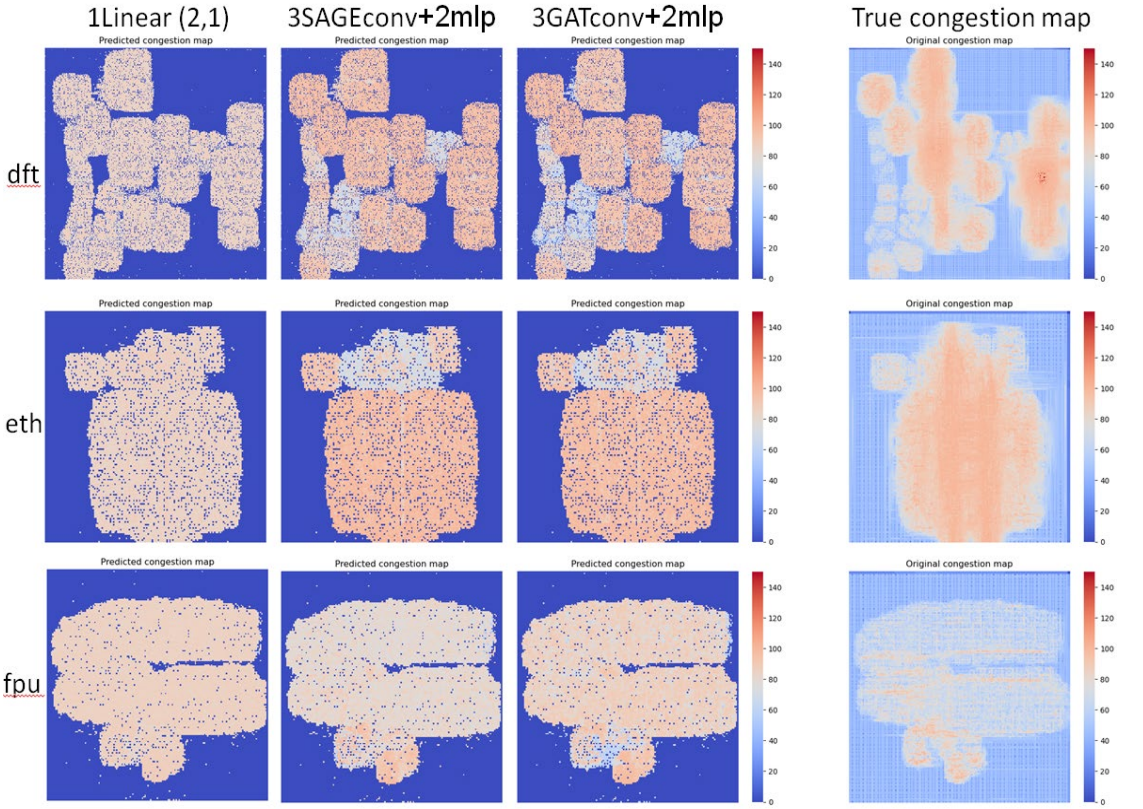

Рис. 2. Карты предсказаний, построенные на основании предсказаний моделей

4. Заключение

В ходе текущего исследования было обработано 25 графов нетлистов, общее количество вершин в которых достигает приблизительно 382 тысячи, используется 753 библиотечных элемента (DEF). Одно из наблюдений заключается в том, что даже без применения обучения, простое предсказание на основе среднего значения может обеспечить абсолютную ошибку в пределах от 10 до 18 по всем графам. Основной проблемой является выбор способа разделения данных на обучающую и тестовую выборки. Некоторые графы значительно различаются как по значению разводимости, так и по числу вершин и используемым элементам. Графы, которые сильно отличаются, могут негативно влиять на результаты тестирования. Отказ от таких графов в пользу более схожих может значительно улучшить результаты. Сравнение с исследованиями Кирби [3] и Гозе [4] и др. выявило потенциальные направления для дальнейшего развития: во-первых, возможность увеличения обучающей выборки данных в 100 раз и, во-вторых, интеграция входных эмбеддингов, основанных на матричном разложении, или обучаемых эмбеддингов для различных типов ячеек.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2022-0008.

Estimation of Congestion Map in the VLSI Design of Digital Blocks with Graph Neural Network

N. V. Zheludkov, I. M. Karandashev, E. S. Kocheva, M. K. Saibodalov, Z. B. Sokhova, A. A. Umnova

Abstract. This paper considers a solution to the problem of estimating the congestion map in the early stages of VLSI layout design of digital blocks by applying a neural network model of machine learning based on a graph neural network. Early prediction of congestion problems will allow the layout engineer to modify design block characteristics such as floorplan, IP-block`s placement and input-output ports to prevent interconnect routing issues at later stages, thereby reducing the number of CAD runs and overall circuit design runtime. The application of graph neural networks allows to take into account additional information about the connections of elements in the netlist for more accurate prediction.

Keywords: VLSI, layout design, congestion map, machine learning, graph neural networks

Литература

1. [Электронный ресурс]. URL: https://theopenroadproject.org/ (дата обращения: 10.11.2023)

2. A.B. Chowdhury, B. Tan, R. Karri, S. Garg. OpenABC-D: A Large-Scale Dataset for Machine Learning Guided Integrated Circuit Synthesis. (2021). ArXiv preprint (2021) arXiv:2110.11292.

3. R. Kirby, S. Godil, R. Roy, B. Catanzaro. Congestionnet: Routing congestion prediction using deep graph neural networks. «2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)», 2019, pp. 217–222.

4. A. Ghose, V. Zhang, Y. Zhang, D. Li, W. Liu, M. Coates. Generalizable cross-graph embedding for GNN-based congestion prediction. «2021 IEEE/ACM International Conference on Computer Aided Design (ICCAD)», 2021, pp. 1–9.

5. P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio. Graph attention networks. ArXiv preprint (2017) arXiv:1710.10903.

6. W.L. Hamilton, Z. Ying, J. Leskovec. Inductive representation learning on large graphs. «Neural Information Processing Systems», 2017.

7. M. Fey, J.E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. «ICLR Workshop on Representation Learning on Graphs and Manifolds», 2019.

8. P. Kudva, A. Sullivan, W. Dougherty. Metrics for structural logic synthesis. «IEEE/ACM International Conference on Computer Aided Design», 2002, pp. 551−556.

Приложение 1. Характеристики нетлистов собранного датасета (белым цветом отмечены графы из обучающей выборки, серым из тестовой)

netlist_name				**AREA**			**NUM_PINS**			**CONGESTION_VALUE**		
	кол-во узлов	**кол-во рёбер**	**библиотечных элементов**	**min**	**max**	**mean**	**min**	**max**	**mean**	**min**	**max**	**mean**
aes_cipher_top	8 419	104366	112	8.19	65.54	26.07	2	9	4.23	18.18	100.00	90.89
des3	1 432	20550	123	8.19	65.54	26.82	2	9	4.54	46.67	100.00	90.83
dynamic_node_top	5 757	79219	129	8.19	73.73	37.88	2	9	4.70	54.55	136.36	91.94
FIR_filter	951	2838	22	8.19	69.63	58.08	2	6	4.40	13.64	86.36	56.19
i2c_master_top	416	3505	71	8.19	65.54	34.25	2	7	4.27	23.33	95.45	63.75
i_buf	66 279	592928	353	8.19	159.74	26.44	2	9	3.64	22.73	204.55	91.72
idft_top	73 584	984272	162	8.19	114.69	42.82	2	9	4.66	26.67	154.55	82.57
IIR_filter	1 524	4474	27	8.19	69.63	55.31	2	9	4.27	22.73	90.91	56.89
mc_top	2 974	30178	165	8.19	77.82	35.84	2	9	4.34	0.00	100.00	73.09
pci_bridge32	6 572	75934	201	8.19	77.82	41.28	2	9	4.63	27.27	100.00	74.49
pcm_slv_top	173	1473	21	8.19	73.73	39.87	2	7	4.21	27.27	77.27	56.60
sasc_top	256	1786	49	8.19	77.82	37.66	2	7	4.29	26.67	81.82	58.97
sha256	4 369	51316	94	8.19	69.63	37.26	2	9	4.33	53.33	106.67	92.44
simple_spi_top	340	2621	66	8.19	65.54	35.13	2	7	4.23	27.27	86.36	63.07
spi_top	1 261	13255	86	8.19	163.84	28.99	2	12	4.25	31.82	100.00	77.88
tv80s	2 613	29844	143	8.19	69.63	28.38	2	9	4.26	26.67	100.00	80.91
usb_phy	269	1732	60	8.19	98.30	32.11	2	7	3.77	36.36	86.67	59.24
~~vga_enh_top~~	~~33 931~~	~~472958~~	~~198~~	~~8.19~~	~~69.63~~	~~40.16~~	~~2~~	~~9~~	~~4.65~~	~~22.73~~	~~518.18~~	~~140.54~~
wb_conmax_top	15 137	248335	84	8.19	73.73	24.31	2	9	4.47	63.33	181.82	124.43
wb_dma_top	1 531	15291	93	8.19	65.54	34.50	2	9	4.21	0.00	100.00	82.80
ac97_top	4 417	51376	100	8.19	65.54	39.78	2	9	4.44	27.27	100.00	70.43
aes128_core	15 609	158622	170	8.19	65.54	26.71	2	9	4.14	59.09	133.33	95.79
dft_top	73 651	981628	166	8.19	114.69	42.73	2	9	4.65	22.73	172.73	82.24
eth_top	21 377	270700	229	8.19	77.82	40.35	2	9	4.68	33.33	104.55	88.25
fpu	39 541	199528	569	8.19	139.26	24.50	2	9	3.06	26.67	100.00	67.97
ВСЕГО:	**382 383**	**4 398 729**	**753**	**8.19**	**163.84**	**35.89**	**2**	**12.00**	**4.29**	**0.00**	**518.18**	**80.56**

Применимость методов машинного обучения для тестирования моделей микропроцессора

Н. А. Гревцев[1], А. Д. Манеркин[2], П. А. Чибисов[3]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, ngrevcev@cs.niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, manerkin@cs.niisi.ras.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, chibisov@cs.niisi.ras.ru

Аннотация. В статье приведен обзор использования методов машинного обучения для различных направлений функциональной верификации. Рассматривается использование машинного обучения в «pre-silicon» верификации, а именно в имитационном тестировании и верификации при помощи UVM. Приводится обзор в области «post-silicon» верификации. Делается вывод об основных областях применения машинного обучения, а также о возможных будущих направлениях исследований.

Ключевые слова: верификация микропроцессоров, машинное обучение, Deep Learning, UVM

1. Введение

В связи с возрастающей сложностью современных микропроцессоров всё больше времени уделяется их верификации. На нее может приходиться до 70% времени создания процессора. В результате верификация становится крайне ресурсозатратным этапом проектирования. Более того, усложнение микроархитектуры современных цифровых устройств приводит к тому, что анализ результатов верификации вручную становится все менее эффективным, а ручное написание и запуск случайных тестов часто является избыточным.

Разрабатываются все новые методы стандартизации и автоматизации тестового окружения. В 2011 году был выпущен единый верификационный стандарт проверки цифровых схем – UVM (Universal Verification Methodology), позволяющий за короткие промежутки времени настраивать стандартную тестовую среду, и впоследствии многократно ее использовать, к примеру, для различных версий проекта.

В последнее время широкое использование в области создания и тестирования программного обеспечения получили методы машинного обучения. Вслед за областью программного обеспечения проводятся все больше исследований на тему применимости методов машинного обучения в области верификации аппаратного обеспечения. Применение данного набора технологий является перспективным направлением для ускорения, повышения качества и эффективности проверки цифровых схем.

В главе 2.1 данной статьи разбирается применение методов машинного обучения для «pre-silicon» верификации микропроцессоров в целом, в главе 2.2 приведен краткий обзор использования машинного обучения в верификации при помощи UVM, в главе 3 рассмотрено их применение для «post-silicon» верификации. В заключении приведены выводы по результатам научных работ в данной области и применимости машинного обучения для задач верификации.

2. Использование машинного обучения в «pre-silicon» верификации

Стадия pre-silicon верификации, то есть до выпуска разрабатываемого микропроцессора, является основным этапом верификации, так как именно на этом этапе разработка проекта осуществляется параллельно с его верификацией, а стоимость исправления ошибок минимальна. Поэтому большая часть работ, посвященная применению методов машинного обучения в верификации, направлена на различные этапы данной стадии.

2.1. Simulation-based тестирование

Ввиду сложности современных микропроцессоров при их проектировании неизбежно возникают ошибки. Ошибки могут нанести серьезный ущерб и привести к высоким издержкам. При этом их исправление в уже изготовленных микросхемах, как правило, невозможно. Поэтому необходимо выявлять как можно больше ошибок перед изготовлением процессора, еще на этапе проектирования.

Для этого используют различные методы функциональной верификации моделей микропроцессоров, нацеленные на обнаружение ошибок, то есть отклонений поведения модели от требуемого. Наиболее широко используемый на практике метод верификации микропроцессоров – имитационное тестирование (simulation-based

verification). Он состоит в программной имитации работы микропроцессора, описываемого моделью, с помощью симулятора в рамках ряда специально разработанных тестовых сценариев, представляющих основные варианты использования функций этого микропроцессора, возможные при его эксплуатации. Другим методом является формальная верификация (*formal verification*). Этот способ заключается в формальном доказательстве соответствия модели микропроцессора заданному набору свойств при всевозможных сценариях взаимодействия с ним. Оба метода имеют как достоинства, так и недостатки, и нередко для верификации микропроцессора используют как тестирование, так и формальную верификацию [1].

Исходя из теоретических предпосылок, случайные воздействия способны реализовать все возможные комбинации при наличии достаточного времени, но на практике при верификации очень сложных устройств случайный подход имеет трудности в своевременной реализации всех комбинаций. В результате, часто требуется направлять тестовую среду с целью реализации конкретных состояний. Такой ограниченно-случайный подход является достаточно мощным средством, однако он опирается на обширный опыт управления тестовой средой для достаточной верификации проекта. С увеличением сложности устройств, управление средой становится все более трудоемким и затратным по времени. Это приводит к тому, что верификация, способная охватить все точки покрытия, становится главным фактором, влияющим на длительность создания устройства [2].

Машинное обучение может значительно улучшить средства достижения необходимого покрытия, предоставляя механизм, с помощью которого можно отслеживать уровень покрытия, управляемые параметры верификации, а также изучать и затем улучшать взаимодействие между ними.

Предложенный алгоритм заключаются в следующем:

1) Для проекта выполняется ряд симуляций, в которых параметры моделирования тестируемого устройства, теоретически способные повлиять на покрытие, изменяются случайным образом.

2) Для каждого отдельного запуска моделирования регистрируется следующий набор входных данных верификации:

- параметры верификации, управляющие входными параметрами;
- параметры тестовой среды;
- параметры конфигурации устройства;
- результаты покрытия, в которых перечислены все состояния функционального покрытия и указано, было ли конкретное состояние достигнуто или нет во время моделирования.

3) Результаты всех симуляций затем обрабатываются алгоритмом машинного обучения, который отслеживает результаты покрытия в зависимости от входных данных.

4) В ходе серии симуляций алгоритм машинного обучения узнает, какие комбинации подмножеств входных данных проверки необходимы для достижения каждого состояния функционального покрытия. Список состояний функционального покрытия, отслеживаемых алгоритмом ML (*Machine Learning*), может быть отфильтрован с целью исключить регулярно встречающиеся состояния функционального покрытия, чтобы сосредоточиться на чрезвычайно редких и труднодостижимых случаях.

5) После обучения алгоритм машинного обучения генерирует набор рекомендаций для входных данных, тем самым увеличивая вероятность попадания в непокрытые состояния, ускоряя процесс разработки проекта.

Одним из примеров реализации подхода является работа [3], которая делает акцент на создание эффективных наборов входных данных с целью увеличения покрытия. Предлагается основанный на ошибке реконструкции метод анализа особенностей наборов данных с помощью глубокого обучения и нахождения таких наборов данных, которые затрагивают труднодостижимые точки покрытия. Экспериментальные результаты показывают, что предлагаемый метод является эффективным в нахождении эффективных наборов данных для моделирования.

Важным направлением работ является сравнение различных методов машинного обучения [4]. Автор работы предлагает новую методологию создания модели с использованием SVM (машины опорных векторов), классификатора повышения градиента и нейронных сетей, нацеленную на замену генерации случайных тестов ускоренным сбором покрытия. При использовании данной методологии авторам удалось добиться достижения целевых показателей покрытия при помощи меньшего количества тестов. Также было обнаружено, что нейронные сети обеспечивают лучшие результаты по сравнению с классификатором повышения градиента и машиной опорных векторов.

Многие авторы фокусируются на использовании конкретных подходов для решения задачи генерации тестов и воздействий. Авторы работы [5], полагая генетические алгоритмы лучшим способом генерации последовательностей входных данных, способных достичь желаемого уровня охвата, разработали и протестировали с использованием трех различных проектов не-

сколько подходов, базирующихся на генетических алгоритмах. Во всех ситуациях процент наборов воздействий, сгенерированных с использованием высокопроизводительных генетических алгоритмов, был выше значений, полученных при использовании случайного моделирования. Вдобавок, в большинстве случаев подход на основе генетических алгоритмов достигал большего значения покрытия за тест в сравнении с результатами случайного моделирования. Эксперименты подтвердили, что в большинстве случаев генетические алгоритмы способны превзойти случайную генерацию воздействий, которая используется в классическом способе проведения верификации, учитывая заполнение уровня покрытия за один тест.

Существуют также исследования, развивающие подходы на основе обучения с подкреплением. Обучение с подкреплением является перспективным методом машинного обучения, способным самостоятельно адаптироваться, применяется для верификации широкого спектра устройств. Работа [6], основываясь на более ранних исследованиях в данной области, предлагает универсальный метод, который уменьшает усилия по созданию функциональных тестов, предлагая возможность их повторного использования для верификации различных RTL-проектов. Новая методика использует обучение с подкреплением при помощи модели «актер-критик» для создания тестовых наборов данных для различных RTL-проектов. Были проведены эксперименты, сравнивающие подход на основе модели “актер-критик” с подходом, когда тестовая система не ограничивает случайные воздействия.

Результаты экспериментов показали, что предложенный метод позволяет увеличить долю сработавших точек покрытия на величину от 11,4% до 50% в зависимости от тестируемого устройства. В случае, если оба подхода достигали 100% покрытия, подход на основе модели «актер-критик» позволял значительно уменьшить время достижения 100% уровня покрытия.

Достаточно много работ посвящено созданию подходов и методологий на основе машинного обучения, оптимизирующих процесс верификации. В диссертации [7] функциональная верификация была разделена на три этапа: планирование и создание тестов, их выполнение и обнаружение ошибок, а также разбор и классификация найденных ошибок. Для решения проблем в рамках этих трех этапов был предложен подход автоматизации, упорядочивания и приближения. В рамках данного подхода, на первом этапе определяется приоритетность определенных аспектов работы, затем автоматизируются действия, относящиеся к приоритетным аспектам, при этом используются приближения, которые снижают точность ради значительного выигрыша в эффективности.

Для эффективного планирования и создания тестов современных систем на кристалле, разработан автоматизированный процесс обнаружения высокоприоритетных аспектов верификации устройства. Кроме того, стало возможным создание более емких тестов, которые оказались до 11 раз более компактными, чем до применения данного процесса.

Для решения проблем в области выполнения тестов и обнаружения ошибок разработана группа решений, которые делают возможным внедрение автоматических и надежных механизмов для обнаружения недостатков устройства в процессе высокоскоростной функциональной верификации. Жертвуя точностью в пользу скорости, данные решения дают возможность выпускать платформы функциональной верификации, которые на более чем на три порядка быстрее традиционных платформ находят ошибки проектирования, которые в противном случае было бы невозможно обнаружить.

Проблемы в области диагностики ошибок решаются при помощи процесса, который полностью автоматизирует выслеживание дефектных компонентов устройства после обнаружения ошибки. Решение, которое обнаруживает дефектные устройства с более чем 70% точностью, значительно сокращает усилия по диагностике для каждой обнаруженной ошибки.

Ряд решений, использующих анализ данных, позволяет снижать трудозатраты на верификацию. В диссертации [8] предлагаются три методологии обучения на основе анализа данных для помощи инженерам-верификаторам в принятии одного из возможных решений (запустить больше тестов, усовершенствовать тестовый шаблон или же сменить его на новый) для достижения целей по функциональному покрытию.

Первая методология определяет важные тесты перед симуляцией, основываясь на том, что уже было промоделировано. Запуская только эти тесты и отбрасывая избыточные, можно сэкономить огромные ресурсы, такие как циклы симуляции. Вторая методология извлекает уникальные свойства из этих важных тестов, определенные при моделировании, и использует их для доработки тестового шаблона. Используя извлеченные сведения, создается больше тестов, схожих с важными. Созданные таким образом тесты более вероятно повышают уровень покрытия, которого было бы тяжелее достигнуть в ином случае. Третья методология анализирует набор существующих тестовых объектов (тестовых шаблонов) и определяет возможное дополнение плана тестирования. Автоматическое добавле-

ние новых тестовых объектов при помощи обучения на основе анализа данных, значительно снижает потребность в ручном труде инженеров-верификаторов по закрытию пробелов в тестовом плане покрытия.

Предлагаемые методологии обучения на основе анализа данных были разработаны и применены для верификации коммерческих микропроцессоров и устройств на платформе СнК. Результаты экспериментов демонстрируют осуществимость и эффективность построения систем анализа данных для повышения скорости и качества верификации.

Важным направлением является создание подходов, решающих проблему выбора тестов и минимизации тестовых наборов для достижения того или иного уровня покрытия. Особенно это актуально для регрессионного тестирования, так как позволяет экономить как время, так и вычислительные мощности. Одной из работ в этой области является диссертация [9], которая уделяет особое внимание использованию машинного обучения для снижения затрат времени на достижения целевых показателей покрытия, обычно занимающего наибольшую часть всего времени верификации. Для этой цели в двух различных экспериментах использовалось как глубокое обучение (Deep Q-Learning), так и обучение с подкреплением. С одной стороны, нейронные сети использовались для помощи с визуализацией, показывающей, стоит ли использовать тот или иной набор воздействий для моделирования, предсказывая уровень покрытия, который он создаст. С другой стороны, использовалось обучение с использованием функции полезности для предсказания минимального набора тестов, необходимого для достижения некоего уровня покрытия кода, путем оптимизации и уменьшения набора тестов, при котором будет достигаться все тот же уровень покрытия.

Результаты этих экспериментов показывают, что среднеквадратичная ошибка модели нейронной сети составляла от 3 до 5 единиц в предсказании двух различных величин покрытия соответственно, что является достаточно хорошим показателем для обучения на маленьком наборе данных. Во-вторых, Q-агент показал результаты на 43% лучше, чем утилита ранжирования покрытия инструмента моделирования (Questa RANKTEST). Применение данного метода может значительно уменьшить требуемое число моделирований в еженедельных регрессиях, что приведет к значительной экономии во времени и ресурсах.

Два вышеописанных подхода позволяют значительно сократить затрачиваемые командой инженеров-верификаторов ресурсы путем ускорения процесса верификации и его автоматизации.

В работе [10] предложена система Intelligent Test Selection (ITS), которая реализует онлайн-подход машинного обучения для обеспечения гибкого решения проблемы выбора тестов. ITS использует машинное обучение для оценки вероятности того, что тест обнаружит ошибку, вызванную конкретным изменением кода. Эти оценки используются для создания уменьшенного набора тестов, которые могут быть использованы для обнаружения скрытых ошибок по причине изменений кода с высокой степенью уверенности. Основные положения данной работы заключаются в следующем:

1) Детерминированный выбор тестов и способ определения приоритетов переопределены в вероятностные.

2) Единая схема для определения и отслеживания RTL-кода и тестов (направленных и случайных) в сложных компиляционных и функциональных потоках проекта.

3) Секционированная многоуровневая архитектура машинного обучения для оценки вероятности ошибки теста для данного изменения RTL-кода или тестов.

4) Динамический выбор функций и онлайн-метод обучения, который позволяет пользователям минимизировать затраты на сбор данных и поддерживать высокое качество результатов.

5) Эффективность подхода демонстрируется на пяти больших коммерческих устройствах и одном с открытым исходным кодом с различными размерами, сложностью и тестовыми методологиями.

Результаты экспериментов подтверждают полезность этого метода для различных стилей проектирования и тестирования. Данный метод с высокой точностью детектирует сбои и одновременно обеспечивает сокращение рекомендуемых наборов тестов для регрессий в 1,3 - 10 раз. Предлагаемый подход имеет ряд ограничений:

1) Сильная зависимость качества результатов от качества данных.

2) Низкий уровень качества результатов для ограниченно-случайных тестов.

Существуют работы, затрагивающие применение машинного обучения в более узконаправленных областях верификации. Одной из таких работ является статья [11], которая рассматривает область межуровневой верификации процессоров. В данной работе предлагается реализация случайного генератора, который на основании наблюдаемой информации о покрытии создает неограниченный поток инструкций, динамически развивающийся во время выполнения. В качестве эталонной модели в условиях тесной ко-симуляции используется ISS (эмулятор набора команд - исполняемая абстрактная мо-

дель процессорного ядра, как правило реализованная на C++). Информация о покрытии постоянно обновляется, основываясь на состоянии выполнения ISS. Подход применяет новую концепцию устаревания на основе покрытия для сглаживания распределения покрытия рандомизированного потока инструкций в течение времени. В сочетании это дает возможность для широкого и глубокого покрытия и обнаружения сложных ошибок в пограничных случаях (corner-case) в RTL-коде ядра.

Эксперименты с 32-битным конвейерным ядром RISC-V серии MINRES The Good Core (TGC) демонстрируют эффективность данного подхода. Было достигнуто намного более равномерное распределение покрытия случайного потока команд при помощи устаревания на основе покрытия, а также была обнаружена сложная микроархитектурная ошибка во взаимодействии уже тщательно протестированного индустриального процессора с прилагаемой инфраструктурой тестовой системы.

2.2. Использование машинного обучения в верификации с использованием UVM

UVM – стандартизированная методология, предназначенная для верификации цифровых устройств. UVM включает в себя библиотеку классов, что позволяет значительно ускорить процесс построения тестового окружения, а также переносить созданные компоненты окружения для верификации различных проектов.

Использование технологий машинного обучения в сочетании с UVM позволяет автоматизировать генерацию тестов и воздействий, что в теории позволяет значительно повысить скорость и качество верификации цифровых схем.

В исследовании [12] для этих целей использовалась трехслойная искусственная нейронная сеть. Входной слой был построен из 32 нейронов, скрытый и выходной – из 128. В качестве функции активации использовалась ReLU (Rectified Linear Unit). Целью внедрения являлось достижения требуемых показателей покрытия утверждений. В качестве DUT (Device under Test) служил простой ЦП. Использование нейронной сети для генерации позволило исключить до 40,2% ранее генерируемых стимулов. Время моделирования сократилось в 24,5 раза, а покрытие утверждений повысилось в 4-29 раз.

Недавние исследования в сфере машинного обучения часто фокусируются на «глубоком обучении» (*deep learning*). Подходы глубокого обучения делают сложные и очень точные выводы из массивных наборов данных. Глубокое обучение требует затратного по вычислительным мощностям процесса обучения и больших по сравнению с традиционным машинным обучением наборов данных, однако оно может обучаться высокоточным моделям, извлекать особенности и соответствия между данными автоматически и применять модели в разных приложениях. Исходя из возможностей глубокого обучения, в работе [13] было предложено использовать функции языка e в сочетании с UVM для обучения глубинной нейронной сети на основе результатов покрытия тестируемого процессора. Покрытие динамически считывалось с помощью Specman coverage API. Для обучения нейронной сети было запущено множество тестов в регрессионном режиме. Тестовое окружение поддерживает поэтапное выполнение, что позволяет в любой момент времени останавливать симуляцию, вводить новые собранные данные и повторно запускать ее. При использовании данного подхода время симуляции сократилось на 33%.

В работе [14] было произведено сравнение различных методов машинного обучения в рамках их применимости для генерации стимулов при верификации кэш-памяти четырехъядерного процессора. Особое внимание было уделено методам опорных векторов, глубинным нейронным сетям и ансамблю решающих деревьев. Для построения моделей машинного обучения использовались библиотеки Scikit-Learn и Keras языка Python. Тестовая среда была построена с помощью UVM. Результаты экспериментов показали, что машинное обучение может уменьшить суммарное время верификации на 70% даже с учетом времени обучения модели. Наиболее эффективным оказался ансамбль решающих деревьев — сокращение времени симуляции на 78%, в то время как метод опорных векторов и глубинные нейронные сети достигли сокращения на 69% и 77% соответственно. Предложенная методология обеспечивает полностью автоматический процесс, что позволяет избежать дорогостоящего ручного труда инженеров-верификаторов, требуемого при разработке направленных случайных тестов.

Также перспективными являются подходы, основанные на применении рекуррентных нейронных сетей (*RNN*), являющихся гибким инструментом, способным эффективно обрабатывать упорядоченные последовательности воздействий. Авторы статьи [15] предлагают новый подход к автоматизации верификации на основе покрытия при помощи оптимизатора на основе рекуррентной нейронной сети. Тестируемыми устройствами служили два процессора, Codasip uRISC и Codix Cobalt, заметно отличающиеся друг от друга уровнем сложности архитектуры. Для Codasip uRISC нейронная сеть состояла из

41 нейрона, для Codix Cobalt — из 1020. В результате применения данного подхода достижение 85% уровня покрытия потребовало примерно на 30% меньше времени, чем при классическом подходе без участия нейронных сетей или любого другого алгоритма оптимизации обработки обратной связи по покрытию.

Машинное обучение в сочетании с UVM также находит применение в области обнаружения и локализации ошибок. В работе [16] был разработан прототип инструмента, который применяет различные алгоритмы машинного обучения для кластеризации и классификации ошибок, обнаруженных в результате тестирования. В качестве входных данных использовались логфайлы симуляции тестовой системы UVM. Данные файлы были предварительно обработаны для создания функций, подходящих для алгоритмов машинного обучения.

Результаты экспериментов показали, что машинное обучение может быть эффективно для классификации первопричин неудачного прохождения тестов в системе UVM. Наиболее эффективным алгоритмом оказался метод «случайного леса» (random forest), обладающий точностью 0,907 и F1-мерой (среднее гармоническое значение точности и полноты) 0,913.

Применение машинного обучения для решения проблемы кластеризации неудачных тестов оказалось менее эффективным. Наибольшую эффективность продемонстрировал плотностный алгоритм кластеризации пространственных данных с присутствием шума в сочетании с методом главных компонент для уменьшения размерности. Показатель AMI (скорректированная взаимная информация) имеет значение 0,593, ARI (скорректированный индекс Рэнда) равен 0,545.

Данные результаты демонстрируют, что алгоритмы кластеризации могут быть недостаточно точными, чтобы на них полностью полагаться. Однако при использовании в сочетании с визуализацией алгоритмы кластеризации могут дать общее представление о том, какие неудачные прохождения тестов вызваны общей причиной.

3. Использование машинного обучения в «post-silicon» верификации

На сегодняшний день post-silicon верификация представляет собой в основном ручной, специализированный процесс. Начиная с первых прототипов микропроцессора, тестовые образцы подключаются к проверочной платформе, которая выполняет большие объемы тестов на высокой скорости. Результаты тестирования проверяются по эталонной модели или содержат самопроверку. До тех пор, пока результаты тестов совпадают, тестирование продолжается, в случае несовпадения результатов, выявляется сбой и начинается ручная отладка. Сначала необходимо воспроизвести сбой, что само по себе является проблемой для ошибок, чувствительных к малозаметным изменениям внутри кристалла. При возникновении таких ошибок, различные выполнения одного и того же теста дают разные результаты: одни проходят успешно, другие — нет. Зачастую труднее всего добиться именно неудачных выполнений.

Ввиду специфики данного этапа верификации и сложности процесса отладки, исследователи в этой области используют подходы машинного обучения для анализа возникших ошибок. К примеру, в статье [17] представлен подход к диагностике ошибок после изготовления кристалла, основанный на машинном обучении. Основанный на методе обнаружения аномалий алгоритм строит модель корректной активности сигнала на кристалле на основе прохождения тестов. Представленный алгоритм применяет методы обнаружения аномалий, схожие с используемыми для выявления мошенничества с кредитными картами, для обнаружения приблизительного цикла возникновения ошибки и набора возможных сигналов, являющихся причиной ошибки.

В сравнении с другими новейшими решениями в этой сфере, данный подход может определить время возникновения ошибки с примерно в 4 раза большей точностью при применении к сложному микропроцессору OpenSPARC T2.

Работа [18] предлагает использование технологий машинного обучения для автоматизации диагностики трассировочных данных устройства, а также для локализации ошибок в процессе посткремниевой верификации. Представленный набор инструментов позволяет создать алгоритм выбора сигналов, который определяет, какие сигналы отслеживать в процессе работы устройства. Выбор сигналов зависит от их типов, а также от связей между ними. При запуске тестируемого устройства трассировочные данные сохраняются для автономного анализа. Техника обработки больших данных, названная Map-Reduce, используется для преодоления проблем обработки огромного массива диагностических данных, полученных от запущенного на ПЛИС-прототипе устройства. Метод кластеризации k-средних применяется для группирования схожих сегментов данных диагностики и идентификации тех, которые встречаются редко во время работы устройства. Вдобавок, предложен набор инструментов для локализации ошибок, в котором кластеризация X-средних исполь-

зуется для группирования пройденных регрессионных тестов на кластеры так, что тесты с ошибками могут быть обнаружены, когда их не удается назначить ни одному их обученных кластеров. Экспериментальные результаты демонстрируют осуществимость предлагаемого подхода при устранении ошибок применительно к группе индустриальных устройств. Применение описанного метода позволяет обнаруживать дефекты устройств при помощи тестирования на основе мутаций.

4. Заключение

В данной статье был приведен краткий обзор использования методов машинного обучения при верификации цифровых устройств.

Наиболее широкое применение нашли данные методы в «pre-silicon» верификации, где машинное обучение в основном используется для генерации эффективных тестов и воздействий. Отдельного упоминания заслуживает использование машинного обучения в сочетании с UVM, которое позволяет достичь весьма значительного сокращения времени симуляции. В то же время достаточно мало работ было посвящено анализу данных на этапе «pre-silicon» верификации с целью автоматизации диагностики и локализации ошибок, а также оптимизации имеющихся тестовых наборов при сохранении уровня покрытия. Направление, посвященное анализу лог-файлов, является достаточно активно развивающимся в области тестирования программного обеспечения, наработки из данной сферы могли бы быть применены и для верификации микропроцессоров.

Машинное обучение также нашло применение в области «post-silicon» верификации, где оно используется для диагностики и локализации сбоев. Количество работ по данному направлению относительно невелико, что также делает его перспективной для исследования областью.

Проведенное авторами исследование применимости методов машинного обучения в верификации микропроцессоров и их моделей позволило определить направление дальнейших работ по внедрению этих инструментов в маршрут разработки современной элементной базы, принятый в ФГУ ФНЦ НИИСИ РАН. Наиболее перспективным направлением, согласно полученным в ходе работ результатом, является применение машинного обучения в сочетании с UVM. Также, эта тематика недостаточно раскрыта в работах российских ученых, следовательно, дальнейшие научные изыскания авторов будут направлены в эту область.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме № FNEF-2022-0004.

A Survey of the Machine Learning Methods Applicability for Microprocessor Models Verification

A. D. Manerkin, N. A. Grevtsev, P. A. Chibisov

Abstract. The article provides an overview about the using machine learning methods in various areas of functional verification. We consider the use of machine learning in "pre-silicon" verification, precisely in simulation-based verification and Universal Verification Methodology. Then we discuss the field of "post-silicon" verification. The main decision was made in conclusion about the main areas of machine learning applications, as well as possible future directions of research.

Keywords: microprocessor verification, machine learning, Deep Learning, UVM

Литература

1. Камкин, А. С. Метод автоматизации имитационного тестирования микропроцессоров с конвейерной архитектурой на основе формальных спецификаций, диссертация на соискание ученой степени кандидата физико-математических наук; Федеральное государственное бюджетное учреждение науки Институт системного программирования им. В. П. Иванникова Российской академии наук. — Москва, 2009. — 181 с.

2. Hughes, William & Srinivasan, Sandeep & Suvarna, Rohit & Kulkarni, Maithilee. (2019). Optimizing Design Verification using Machine Learning: Doing better than Random.

3. K. H. Mami Miyamoto, Finding effective simulation patterns for coverage-driven verification using deep learning, in: SASIMI 2016 Proceedings, 2016, pp. 335–340.

4. A.S. Jagadeesh Accelerating coverage closure for hardware verification using machine learning, Master thesis, 2019, Texas A&M University

5. Danciu, Gabriel Mihail, and Alexandru Dinu. 2022. "Coverage Fulfillment Automation in Hardware Functional Verification Using Genetic Algorithms" Applied Sciences 12, no. 3: 1559. https://doi.org/10.3390/app12031559

6. S. L. Tweehuysen, G. L. A. Adriaans and M. Gomony, "Stimuli Generation for IC Design Verification using Reinforcement Learning with an Actor-Critic Model," 2023 IEEE European Test Symposium (ETS), Venezia, Italy, 2023, pp. 1-4, doi: 10.1109/ETS56758.2023.10174129.

7. Mammo, Biruk. (2017). Reining in the Functional Verification of Complex Processor Designs with Automation, Prioritization, and Approximation.

8. Chen, W. (2014). Data Learning Methodologies for Improving the Efficiency of Constrained Random Verification. UC Santa Barbara. ProQuest ID: Chen_ucsb_0035D_12213. Merritt ID: ark:/13030/m5sv2nkc. Retrieved from https://escholarship.org/uc/item/0db4j3jp

9. Aggoune M. (2022) Acceleration of Hardware Code Coverage Closure Using Machine Learning. University of Oulu, Degree Programme in Computer Science and Engineering, 56 p.

10. G. Parthasarathy et al., "RTL Regression Test Selection using Machine Learning," 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC), Taipei, Taiwan, 2022, pp. 281-287, doi: 10.1109/ASP-DAC52403.2022.9712550.

11. N. Bruns, V. Herdt, E. Jentzsch and R. Drechsler, "Cross-Level Processor Verification via Endless Randomized Instruction Stream Generation with Coverage-guided Aging," 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 2022, pp. 1123-1126, doi: 10.23919/DATE54114.2022.9774771.

12. Wang, F.; Zhu, H.; Popli, P.; Xiao, Y.; Bodgan, P.; Nazarian, S. Accelerating Coverage Directed Test Generation for Functional Verification: A Neural Network-Based Framework. In Proceedings of the Great Lakes Symposium on VLSI, ACM, New York, NY, USA, 23–25 May 2018; pp. 207–212.

13. Dinu, A.; Ogrutan, P.L. Opportunities of using artificial intelligence in hardware verification. In Proceedings of the 2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME), Cluj-Napoca, Romania, 23 October 2019; pp. 224–227.

14. Gogri, S.; Hu, J.; Tyagi, A.; Quinn, M.; Ramachandran, S.; Batool, F.; Jagadeesh, A. Machine Learning-Guided Stimulus Generation for Functional Verification. In Proceedings of the Design and Verification Conference (DVCON-USA), Virtual Conference, 2–5 March 2020.

15. Fajcik, M.; Smrz, P.; Zachariasova, M. Automation of Processor Verification Using Recurrent Neural Networks. In Proceedings of the 2017 18th International Workshop on Microprocessor and SOC Test and Verification (MTV), Austin, TX, USA, 11–12 December 2017; pp. 15–20.

16. Truong, A.; Hellstrom, D.; Duque, H.; Viklund, L. Clustering and Classification of UVM Test Failures Using Machine Learning Techniques. In Proceedings of the Design and Verification Conference (DVCON), San Jose, CA, USA, 26 February–1 March 2018.

17. A. DeOrio, Q. Li, M. Burgess and V. Bertacco, "Machine learning-based anomaly detection for post-silicon bug diagnosis," 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 2013, pp. 491-496, doi: 10.7873/DATE.2013.112.

18. Mandouh, Ema & Wassal, Amr. Application of Machine Learning Techniques in Post-Silicon Debugging and Bug Localization. Journal of Electronic Testing, 2018, 34. 1-19. 10.1007/s10836-018-5716-y.

Оценки вероятности промаха при случайном тестировании кэша

А. С. Куцаев[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, koutsaev@niisi.msk.ru

Аннотация. Вероятность промаха при случайном тестировании системы кэшей зависит в основном от распределения памяти. Оно позволяет ограничить число активных строк, создавая тем самым нагрузку на кэш. Перебор областей памяти в генераторе тестов позволяет сократить вспомогательные действия в тесте, не связанные непосредственно с тестированием. Оценки вероятности промаха в кэшах первого и второго уровня при различных условиях позволяют выбрать параметры для генерации эффективных тестов.

Ключевые слова: случайные тесты, вероятность промаха в кэше, перебор областей памяти

1. Введение

Тестирование системы кэшей и TLB с помощью генератора случайных тестов [1, 2] требует подбора управляющих параметров с учетом задачи, особенно в части случайного выбора адресов памяти. Случайный выбор адресов без дополнительных условий обычно приводит к неэффективным тестам. Причина в том, что размер кэшей достаточно велик, и случайные обращения к памяти вызывают в кэше в основном промахи без замещения данных.

Для тестирования интересен режим частых попаданий, когда данные целиком помещаются в кэш, и обмен с памятью происходит наиболее быстро. Также полезен режим частых промахов, создающий большую нагрузку на кэш. В случайном тесте эти режимы могут осложняться добавлением переходов, циклов и исключительных ситуаций. Переходный режим случайного заполнения кэша менее интересен, так как не связан с определенным видом нагрузки на кэш. В генераторе тестов есть средство быстрого случайного заполнения кэша.

Для создания эффективных тестов кэшей наиболее важен выбор распределения памяти. Он позволяет реализовать т.н. режим "горячих строк", в котором нагрузка приходится на небольшое подмножество строк кэша. Далее для получения "горячих строк" отводимая память состоит из небольших одинаковых областей, адреса начала которых выровнены по размеру множества кэша.

Режим частых попаданий проще, так как требует только, чтобы число областей памяти N не превышало числа множеств кэша A. Для режима частых промахов условия $N > A$ мало. При случайном выборе областей памяти вероятность промаха равна $1 - A/N$, так что число областей должно составлять несколько десятков. При тестировании кэша 1-го уровня области можно расположить так, что регистры базы адреса не будут требовать частой перезагрузки. Но для кэша 2-го уровня на каждую область требуется отдельный регистр, так что перезагрузки неизбежны. Еще больше областей памяти может потребоваться при тестировании TLB.

Генератор тестов предлагает режим перебора областей памяти, в котором случайный выбор адресов происходит определенное число раз в одной области или группе областей. После этого происходит переход к следующей области или их группе из списка. Список допустимых областей один и тот же для операций загрузки и сохранения. Режим перебора позволяет перезагружать регистры адреса группами по мере необходимости. В результате дополнительные действия, не связанные с целью тестирования, отделены от основных действий.

В главе 2 оценивается вероятность промаха в кэше первого уровня, если обращения к памяти ограничены ее чтением. В главе 3 делается то же самое, если чтение и запись равновероятны. В главе 4 оценивается промаха в кэше второго уровня, также в случае чтения и записи. В главе 5 приведены выводы.

2. Кэш L1, только чтение

Рассмотрим промахи в кэше первого уровня (L1) при переборе областей памяти, если выполняется только чтение данных. Схема оценки с некоторыми изменениями применима и для чтения-записи. Пусть память данных состоит из N одинаковых областей, адреса начала которых выровнены по размеру множества кэша, т.е. каждая область связана с одними и теми же строками кэша. Примем для простоты, что перебор областей происходит по одной. В цикле перебора N шагов, число циклов не ограничено, на каждом шаге выполняется ровно K обращений к памяти, число активных строк кэша L.

Далее используется определенная система

кэшей. В ней кэш L1 со сквозной записью, кэш 2-го уровня (L2) с обратной записью. В вычислительных примерах в кэше L1 4 множества (ways), число строк в множестве 256, длина строки 16 байтов. В кэше L2 4 множества, число строк в множестве 4096, длина строки 32 байта. Строка L2 здесь отвечает двум строкам L1, что может быть полезно в некоторых тестовых ситуациях.

Когда кэш заполнен, каждое чтение памяти приводит к попаданию в кэше либо к промаху с замещением данных, которые не использовались дольше всего. Тег кэша определен областью памяти, т.е. номером шага в цикле перебора, строка кэша определяется младшими битами случайного адреса.

Вероятность того, что случайный адрес будет выбран в определенной строке S кэша, равна $1/L$. При K выборах адреса на шаге вероятность того, что строка S будет выбрана на шаге хотя бы один раз, равна

$$P_1(L,K) = 1 - (1 - 1/L)^K \qquad (2.1)$$

Это дополнение до 1 вероятности, что на этом шаге строка S не будет выбрана ни разу. При чтении памяти тег адреса и данные копируются в кэш на этом шаге, если они не скопированы ранее. Повторный выбор строки S на этом же шаге приводит к попаданию в кэш и не влияет на наличие и очередность тегов в строке.

При длительном выполнении теста можно выбрать область памяти и принять, что цикл перебора начинается с нее. Обозначим T_1 тег адреса, отвечающий области на первом шаге цикла. Оценим вероятность промаха в кэше на первом шаге цикла. Пусть строка S выбрана на первом шаге цикла с номером C+1 (при нескольких выборах строки S на шаге берется первый). Попадание в кэше происходит, если тег T_1 уже есть в строке S, т. е. если в предыдущий раз тег T_1 был выбран в строке S на первом шаге цикла с номером B и с тех пор не был замещен в циклах B, B+1, ... C. Для этого нужно, чтобы в циклах с B по C замещение тега (любого) в строке S произошло менее A раз.

Поскольку кэш заполнен, в строке S все теги действительны. Замещение тега T_1 в ней не может происходить на первом шаге циклов с B по C по условию. Кроме того, замещение тегов в строке S возможно не на всех шагах с 2 по N. Если в начале цикла строка S содержит тег T_X, и этот тег не замещен до шага X, на котором выбирается данный тег, то на шаге X будет попадание в кэше, а не замещение. В каждом цикле число шагов, на которых возможно замещение тега, может принимать значения от N-A до N-1. Для оценок вероятности замещения снизу и сверху нужно использовать число шагов на цикл N-A и N-1, соответственно.

Вероятность выбора строки S на шаге перебора хотя бы один раз равна $P_1(L, K)$ (2.1). Вероятность того, что на m шагах перебора замещение тега произойдет ровно i раз, обозначим G_i. Вероятности G_i. (i = 0, 1,...m) являются членами разложения по степеням бинома $1 = ((1 - P_1) + P_1)^m$.

$$G_i = C_m^i (1 - P_1)^{m-i} P_1^i \qquad (2.2)$$

где C_m^i - биномиальные коэффициенты. Вероятность того, что последний выбранный тег в строке S не будет замещен на m шагах, обозначим $P_T(m)$. Она равна сумме первых A членов разложения бинома. Эта вероятность тем меньше, чем больше шагов участвует в замещении.

$$P_T(m) = \sum_{i=0}^{A-1} G_i \qquad (2.3)$$

До начала цикла C+1 предыдущий выбор тега T_1 в строке S может быть в цикле C, C-1, C-2,... 1, либо тег не выбран ни разу. Это полный набор событий с вероятностями P_1, $P_1(1\text{-}P_1)$, $P_1(1\text{-}P_1)^2$ и т.д.; вероятность невыбора равна $(1\text{-}P_1)^C$. Вероятность того, что предыдущий выбор тега T_1 был в цикле B и тег не замещен до начала цикла C+1 на m шагах, равна произведению $P_1(1\text{-}P_1)^{C\text{-}B}$ на $P_T(m)$. Суммируя для всех циклов от 1 до C и заменяя C-B+1 на i, получим оценки для вероятности незамещения тега T_1:

$$P_{SC}(N) = \sum_{i=1}^{C-B+1} P_1(1 - P_1)^{i-1} P_T(iM) \qquad (2.4)$$

где M=N-A для оценки сверху и M=N-1 для оценки снизу, согласно замечанию при выводе (2.3). С ростом C-B эти суммы сходятся как степенной ряд или быстрее, так что для анализа можно брать их предел $P_S(N)$. Его дополнение до 1 дает оценки для вероятности первого промаха в строке S на первом шаге цикла C+1, когда C достаточно велико.

$$P_F(N) = 1 - \sum_{i=1}^{\infty} P_1(1 - P_1)^{i-1} P_T(iM) \qquad (2.5)$$

Для оценки $P_F(N)$ снизу нужно брать в (2.5) M=N-A.

Можно сравнить $P_F(N)$ с вероятностью промаха при случайном выборе областей. Примем число обращений к памяти на шаге K=1, чтобы исключить влияние повторных выборов (оно рассмотрено далее). На Рис. 1 показаны результаты для случайного выбора областей (1-A/N) и оценки снизу для перебора при L=16 и L=32 в зависимости от N. Здесь видно, что при N>16 вероятность промаха при переборе областей выше, чем при их случайном выборе. Также можно заметить, что при N>16 вероятность промаха падает с ростом числа строк, поскольку замещение тегов на меньшем числе строк происходит более

интенсивно.

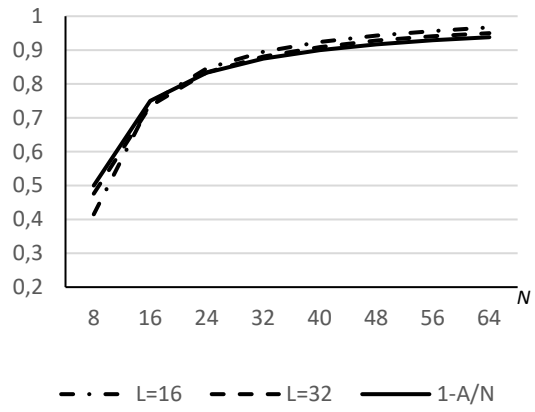

Рис. 1. Вероятности промаха в кэше L1 при *K*=1, только чтение.

При втором и следующих выборах адреса на шаге перебора вероятность промаха зависит от возможности повторного выбора строки. При первом выборе строки на шаге вероятность промаха для нее одна и та же, $P_F(N)$. При повторном выборе будет попадание в кэше.

Оценить вероятности двух и более промахов на шаге можно с помощью рекуррентных соотношений. Очередная новая строка выбирается на шаге с вероятностью 1 - $h{\cdot}p$, где $p = 1/L$, а h - число уже выбранных на шаге новых строк. Обозначим вероятность выбора h новых строк при K выборах через $P_{K-1}^{h}(p)$; это полином степени K-1 от p. Для K=2 имеем $P_1^1(p) = p,\ P_1^2(p) = 1 - p$.

Используя вероятность выбора очередной новой строки 1-$h{\cdot}p$, получим для K выборов:

$$P_K^{K+1}(p) = P_{K-1}^{K}(p) \cdot (1 - Kp)$$

$$...P_K^h(p) = P_{K-1}^{h-1}(p) \cdot (1 - (h-1)p) + + P_{K-1}^{h}(p) \cdot hp, \qquad h = 2, \dots, K$$

$$P_K^1(p) = P_{K-1}^1(p) \cdot p$$

Среднее число новых строк, нормированное на K, дается выражением:

$$M_K(p) = \frac{1}{K}\sum_{h=1}^{K+1} i \cdot P_K^h(p) \qquad (2.6)$$

Так как в случаях, представляющих интерес, значение $p = 1/L$ невелико, для вычисления $M_K(p)$ в них достаточно нулевой и первой степени p:

$$M_K(p) = 1 - \frac{K}{2}p + \cdots \qquad (2.7)$$

Так, для L=16 и L=32 зависимость $M_K(p)$ от K выглядит практически линейной. Коэффициент при p^2 зависит от K более сложно. Среднее число промахов на шаге перебора дается выражением $K \cdot M_K(p) \cdot P_F(N)$. В принятых условиях увеличение K до 6 - 8 не приводит к заметному снижению эффективности, хотя и не дает роста вероятности промаха.

3. Кэш L1, чтение и запись

Оценки для чтения и записи памяти сложнее, так как обе операции влияют на порядок тегов в строке, но только чтение обеспечивает наличие тега и приводит к замещению тега. Для оценки вероятности замещения вместо $P_1(L,K)$ нужно использовать $P_2(L,K)$. Это вероятность того, что на шаге перебора будет хотя бы одно чтение памяти. Если чтение и запись памяти происходят равновероятно, выражение для $P_2(L,K)$ аналогично (2.1):

$$P_2(L,K) = 1 - (1 - 1/2L)^K \qquad (3.1)$$

Вероятность незамещения тега T_1 на m шагах перебора $P_{TW}(m)$ принимает вид

$$P_{TW}(m) = \sum_{i=0}^{A-1} G_i, \qquad (3.2)$$

где $G_i = C_K^i (1 - P_2)^{K-i} {P_2}^i$. При большом числе циклов вероятность незамещения тега T_1 к началу любого цикла будет близка к предельной, обозначаемой $P_{SW}(N)$. Она находится аналогично вероятности для только чтения $P_S(N)$, с заменой $P_T(m)$ на $P_{TW}(m)$, следующим образом.

Вероятность незамещения тега T_1 до начала цикла C+1 складывается, как и выше (2.4), из вероятностей предыдущего выбора тега T_1 на циклах C, C-1,..., B , умноженных на вероятности незамещения до цикла C+1. Добавляется учет того, что перед записью в память тег T_1 в строке S необходим. Составляющие $P_{TW}(m)$ можно свести в таблицу:

Номер цикла	Вероятность выбора строки	Вероятность незамещения
C	P1	PTW(M)
C-1	(1-P1)P1	PTW(2M)
C-2	(1-P1)2P1	PTW(3M)
...	...	...
B	(1-P1)C-BP1	PTW((C-B+1)•M)

В отличие от случая только чтения появляется еще один сомножитель: вероятность чтения или, при наличии тега T1, записи, при условии выбора строки S. Она одинакова для всех циклов и равна . P_2/P_1 + (1 - P_2/P_1)·$P_{SW}(N)$. Здесь условная вероятность хотя бы одного чтения P_2 делится на вероятность условия, что строка выбрана, т.е. P_1. Вероятность отсутствия чтения (т.е. записи) умножается на вероятность наличия тега T_1 в строке перед началом цикла. Она уже представляет собой предел при C-$B \to \infty$. Пере-

множая и суммируя вероятности, получим в пределе равенство с $P_{SW}(N)$ в левой и правой части, из которого можно выразить искомое:

$$P_{SW}(N) = \frac{P_2 Z}{1-(P_1-P_2)Z}, \qquad (3.3)$$

где $Z = \sum_{i=1}^{\infty}(1-P_1)^{i-1}P_{TW}(iM)$

Вероятность первого промаха при чтении-записи $P_{FW}(N) = 1 - P_{SW}(N)$. Оценки снизу вероятности первого промаха при чтении и записи показаны на Рис. 2. По сравнения со случаем только чтения эти оценки ниже. Причина в том, что при чтении и записи замещение тегов происходит примерно вдвое реже.

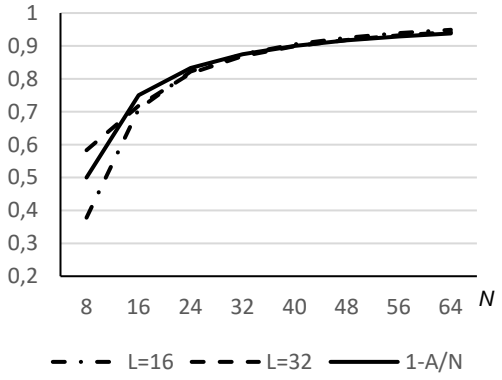

Рис. 2. Вероятности промаха в кэше L1 при K=1, чтение и запись.

При двух и более выборах адреса на шаге перебора вероятность промаха зависит, как и выше, от возможности повторного выбора строки на шаге. Отличие в том, что после промаха при записи повторный выбор строки также приводит к промаху. Здесь вероятность промаха определяет не число строк, выбранных заново, а число строк, выбранных для чтения. Если на шаге уже выбрано для чтения H строк, то вероятность выбора такой строки равна $H{\cdot}p$ (где p=1/L), при этом происходит попадание в кэш. Вероятность выбора строки, где чтения не было, равна 1-$H{\cdot}p$. При этом рост числа промахов M происходит при чтении и записи, а рост H только при чтении. Эти правила можно представить в виде аналога разностной схемы, где по вероятностям промаха $P_K^{H,M}$ для индексов H, M на уровне K вычисляются вероятности $P_{K+1}^{H,M}$ на уровне K+1. Исходно $P_1^{0,1} = P_1^{1,1} = 1/2$, остальные нули. Предварительно $P_{K+1}^{H,M}$ обнуляется, затем для всех индексов:

$$P_{K+1}^{H,M} = P_{K+1}^{H,M} + P_K^{H,M}\, pH$$

$$P_{K+1}^{H,M+1} = P_{K+1}^{H,M+1} + P_K^{H,M}\,(1-pH)/2$$

$$P_{K+1}^{H+1,M+1} = P_{K+1}^{H+1,M+1} + P_K^{H,M}\,(1-pH)/2$$

После получения данных на уровне K, чтобы получить P_K^M, нужно для каждого M просуммировать вероятности $P_K^{H,M}$, по H от 0 до K. Среднее число новых строк, нормированное на K, дается выражением (2.6).

4. Промахи в кэше L2

Для системы кэшей 1-го и 2-го уровня можно использовать набор областей памяти, аналогичный предыдущему. Выравнивание областей памяти должно отвечать размеру множества кэша L2, который значительно больше, чем в L1. Другие возможности организации памяти будут рассмотрены далее. Будем рассматривать чтение и запись памяти с равными вероятностями.

Чтобы в кэше L2 произошел промах, необходимо отсутствие в кэше тега и копии данных для адреса памяти. При записи в память этого достаточно, но при чтении необходим также промах в кэше L1, иначе обращения к L2 не будет. Зависимость условия в L1 и L2 для случая чтения выглядит следующим образом.

Пусть строке S2 кэша L2 соответствует пара строк SA и SB кэша L1, и пусть в строке SA происходит промах при чтении по адресу с тегом T1, т.е. на первом шаге цикла. Перед этим происходит следующая последовательность событий.

- Последнее перед промахом копирование данных по адресу с тегом T1 в строку SA при чтении либо обновление в строке SA при записи в память;
- Одновременно с этим в строке S2 при чтении и записи происходит копирование или обновление соответствующего тега T2 и данных;
- Замещение тега T1 в строке SA в одном или нескольких циклах перебора;
- Замещение тега T2 и данных в строке S2, не позже, чем в L1 (см. ниже).

Теги T_1 и T_2 отвечают адресам одной и той же области памяти, т.е. действуют на одном и том же шаге перебора. Случаев замещения в строке S_2 примерно вчетверо больше, чем в строке S_A, за счет (a) операций записи в память в строке S_A и (b) операций чтения и записи в строке S_B. Таким образом, после замещения в строке S_2 тег T_2 может быть снова прочитан, если позволяют условия (два и более цикла перебора для замещения).

Тогда при промахе в строке S_A возможно попадание в строке S_2. Этот случай маловероятен; в основном промаху в строке S_A отвечает замещение тега T_2 в строке S_2, т.е. промах в L2.

Если же в строке S_A происходит попадание, то тег T_1 и данные не были замещены со времени предыдущего чтения/обновления (как минимум за один цикл перебора). Но так как в кэше L2 случаев замещения в строке S_2 вчетверо больше, то возможно как наличие тега T_2, так и его замещение.

Таким образом, при чтении памяти промах в кэше L1 имеет некоторую положительную корреляцию с замещением соответствующего тега в кэше L2. В случае полной зависимости этих событий вероятность промаха в L2 равна вероятности первого промаха в L1, $P_{FW}(N)$. В случае полной независимости $P_{FW}(N)$ нужно умножить на вероятность замещения соответствующего тега в L2. Между этими крайними случаями данное произведение вероятностей можно рассматривать как оценку снизу для вероятности промаха в L2 при чтении памяти. При записи в память состояние L1 не влияет на вероятность промаха в L2.

Вероятность незамещения соответствующего тега в L2, $P_{S2}(N)$, находится по тем же правилам, что $P_S(N)$ в L1 в случае только чтения (2.4), так как в L2 все операции отражаются в кэше. Число строк L при этом нужно уменьшить вдвое по сравнению с L1.

Оценка снизу вероятности промаха в кэше L2 в начале шага перебора

$$P_{F2(}N) = (1 - P_{S2}(N))\cdot(1 + P_{FW}(N)) /2 \quad (4.1)$$

Зависимость вероятности промаха от N позволяет выбрать приемлемое число областей, на Рис. 2 видно, что это N=32 и выше. Зависимость вероятности промаха от числа строк L кэша позволяет выбрать приемлемый размер области. На Рис. 3 показаны зависимости вероятности промаха $P_{FW}(L)$ для кэша L1 и $P_{F2}(L)$ для кэша L2 при числе областей N=32 и N=64 (число строк везде отвечает размеру строки L1). Здесь видно, что вероятность промаха мало меняется, за исключением очень небольших областей (L=16, размер 128 байтов).

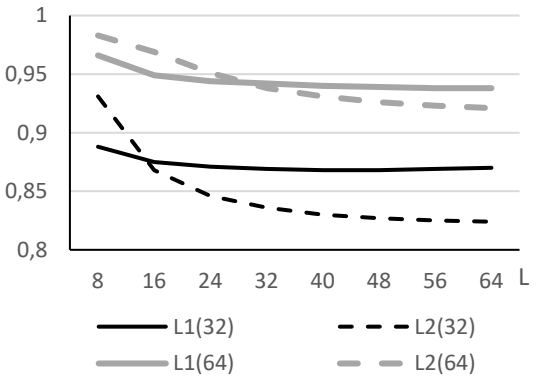

Рис. 3. Вероятности промаха в кэшах L1 и L2 при N=32 и N=64.

Можно также рассмотреть организацию памяти, в которой адреса начала соседних областей вдвое меньше размера множества кэша в L2. Тогда в кэше L1 все остается по-прежнему, а в кэше L2 будет два отрезка активных строк, для областей с четными и нечетными номерами, соответственно. Работа с кэшем L2 для этих групп областей происходит независимо, поэтому в расчетах вероятности промаха нужно брать число областей в шаблоне N, деленное на 2. Уменьшая дальше смещение областей, можно в итоге получить ситуацию, в которой память целиком помещается в кэш L2, и промахов в нем не будет. Выбирая смещение, можно регулировать вероятность промахов в L2, тогда как в L1 она не будет меняться.

5. Заключение

Использование перебора областей памяти при случайном тестировании позволяет использовать большое количество областей памяти без снижения эффективности при тестировании кэшей и TLB. Разработанные способы оценки вероятности промаха сверху и снизу для кэшей первого и второго уровня при переборе областей позволяют выбрать параметры для генерации эффективных случайных тестов.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН «Проведение фундаментальных научных исследований (47 ГП)» по теме № FNEF-2022-0004 «Разработка архитектуры, системных решений и методов для создания микропроцессорных ядер и коммуникационных средств семейства систем на кристалле двойного назначения. 0580-2022-0004», Рег № 122041100063-6.

Miss Probability Estimates for Random Testing of the Cache

A. S. Koutsaev

Abstract. The probability of a miss in a random test of the cache system depends mainly on memory layout. It makes it possible to limit the number of active cache lines, thereby forcing the load on the cache. Enumerating memory areas in the test generator allows one to reduce auxiliary actions in the test that are not directly related to the target of testing. Estimates of the probability of a miss in caches of the first and second levels under various conditions serve to select parameters for generating effective tests.

Keywords: random tests, probability of cache miss, enumeration of memory areas.

Литература

1. А.С. Куцаев. Развитие генератора случайных тестов tergen. Труды НИИСИ РАН 2018, Том 8, № 1, 19-26.

2. А.С. Куцаев, Случайные тесты с перебором классов инструкций. Труды НИИСИ РАН 2022, Том 12, № 1-2, С. 32-37.

Влияние зернистости металлического затвора кремниевых конических GAA нанотранзисторв на флуктуации порогового напряжения

Н.В. Масальский[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, volkov@niisi.ras

Аннотация. Обсуждается влияние зернистости металлического затвора на флуктуацию порогового напряжения кремниевого полевого GAA нанотранзистора. На основе теоремы Пельгорма разработана методика достоверной оценки флуктуации порогового напряжения. В диапазоне длин затворов транзисторов от 11 до 25 нм и средних размеров зерен от 3 до 10 нм получены коэффициенты Пельгорма. Относительные погрешности между модельными значениями стандартного отклонения порогового напряжения и данными полученными из 3D моделирования практически в 95% случаев ниже 5%.

Ключевые слова: перечисление ключевых слов через запятую

1. Введение

Степень влияния различных технологических источников флуктуации на электро-физические характеристики транзистора возрастает по мере применения новых технологических норм и поэтому, получение достоверной оценки этого действия является актуальной задачей [1-3]. Одним из таких факторов флуктуации является зернистость металлического затвора (ЗМЗ) транзистора [2-5]. Прямые исследования флуктуации требуют моделирования большого количества различных конфигураций транзистора, чтобы иметь большую статистическую выборку. Прогноз на основе теоремы Пельгорма позволяет оценить влияния различных факторов на флуктуацию порогового напряжения (σUTh) транзистора любой архитектуры [6]. Этот метод используется для оценки влияния механизма ЗМЗ на флуктуации порогового напряжения кремниевых конических GAA (gate-all-around) нанотранзисторов. Следует уточнить, в коническом транзисторе только рабочая область (канал) имеет форму конуса. При этом, что важно, о стороны истока диаметр больше чем со стороны стока [7]. Выбор такой конструкции обусловлен рядом полезных свойств: всеобъемлющее влияние на дрейф носителей, суперспособность подавлять коротко-канальные эффекты (ККЭ), нивелировать эффекты, связанные с горячими носителями и при определенном (оптимальном) отношении диаметров получать ток стока выше на 15-25% по сравнению с цилиндрической GAA конструкцией. Мы перечислили только главные отличительные свойства конической архитектуры, при этом она сохраняет плюсовые характеристики свойственные всем цилиндрическим GAA нанотранзисторам. Ключевой параметр любого полевого транзистора, и рассматриваемый конический не исключение, пороговое напряжение. Исходя из современных технологических норм в настоящей работе в диапазоне длин затворов транзисторов (Lg) от 11 до 25 нм и размеров зерен (Gs) от 3 до 10 нм и получены коэффициенты Пельгорма, которые позволяют сделать достоверные оценки параметра σUTh «на коленке».

2. Постановка задачи исследования

Для точного анализа влияния ЗМЗ на характеристики конических GAA нанотранзисторов необходимо [5]:

i) использовать 3D моделирование,

ii) моделировать большие ансамбли различных модификаций транзисторов.

Эти два фактора заметно увеличивают вычислительные затраты. Поэтому мы используем подход на основе теоремы Пельгорма. Этот метод эффективен в плане экономии вычислительных ресурсов [4], поскольку, рассчитав коэффициент Пельгорма, можно прогнозировать диапазон параметра σUTh для различных длин и диаметров затвора GAA транзистора. Следует отметить, что есть и другие методы, позволяющие сократить время вычислений, которые основаны на машинном обучении [8, 9] или карта чувствительности к флуктуациям [10, 11].

Классический закон Пелгрома был сформулирован для планарных транзисторных структур [12], который утверждает, что стандартное отклонение порогового напряжения (σUTh) обратно пропорционально эффективной площади

металлического затвора [4]. Этот закон может быть распространен на современные архитектуры, включая кремниевые конические GAA нанотранзисторы, который в данном случае можно представить в виде [2]:

$$\sigma U_{Th} = \frac{A_{Th}}{\sqrt{L_g W_{eff}}}, \qquad (1)$$

где ATh - коэффициент Пельгорма, Weff - эффективный периметра затвора.

В координатах σUTh и зависимость σUTh ( ) предоставляет собой прямую линию (в декартовой системе координат в первом квадранте) наклон которой есть коэффициент Пельгорма. Однако, размерность коэффициента ATh – mV nm – является несистемной, т.е. в некотором роде этот коэффициент лишен физического смысла. Это создает предпосылку для более детального исследования структуры (компонентов) ATh. Следует отметить, что сама линия не достигает начала координат, однако ее продолжение должно проходить через него. Эти свойства делают подход Пельгорма простым и надежным инструментом для оценки влияния флуктуации σUTh [4], в частности, из-за механизма МЗМ. Все исследуемые в рамках нашей работы модификации транзисторов включают широко используемый нитрид-титановый металлический затвор из-за его совместимости с оксидом кремния и низкой емкости [13]. Механизм ЗМЗ проявляется из-за случайной ориентации металлических зерен, которые образуются в процессе формирования металлического затворе [13, 14]. Поэтому, средний размер зерна Gs определяется характеристиками технологических операций, например, время осаждения, температура и длительность отжига. Опираясь на многочисленные работы других исследователей [3, 5, 14, 15] интересующие нас электро- физические характеристики металлических зерен затвора были установлены так: i) две возможные ориентации металлических зерен с работой выхода φМ 4.4 и 4.6 эВ, ii) вероятность их появления 40 и 60%, iii) профили ЗМЗ генерируются с использованием диаграмм Poisson-Voronoi в зависимости от значения параметра Gs. На рис. 1 показана схема металлического затвора GAA нанотранзистора, на котором изучается влияние механизма ЗМЗ.

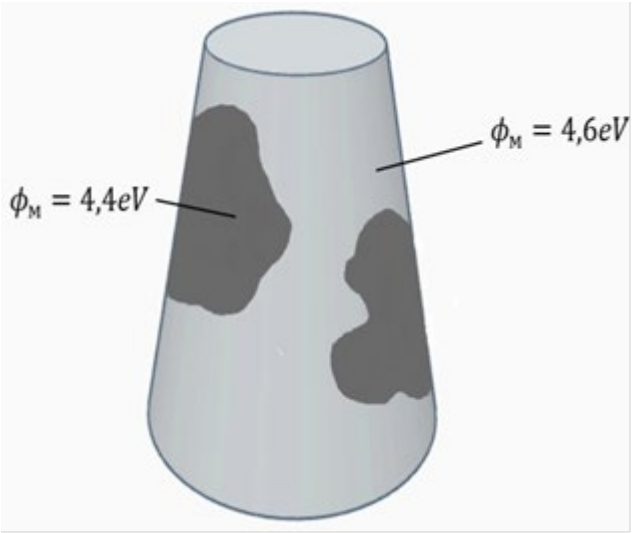

Рис. 1. Схема металлического затвора конического GAA с зернистостью металла

3. Модель ЗМЗ, расчеты и обсуждение

Аналогично предыдущей наше работы [16] для получения оценки полагаем, что механизм ЗМЗ является единственным источником флуктуации σUTh. Поскольку теорема Пельгорма применима для описания флуктуации порогового напряжения из-за разных механизмов [12], то в предположении о линейности зависимости коэффициента Пельгорма от Gs, что также было экспериментально продемонстрировано, следовательно его можно представить таким выражением:

$$A_{Th} = \gamma_G G_s, \qquad (2)$$

где γ_G - коэффициент пропорциональности, размерность которого В, и он также может быть подгоночным параметром. При этом коэффициент γ_G не зависит от управляющих напряжений на транзисторе [12]. Однако он будет зависеть от металла, используемого в процессе формирования затвора [17]. Очевидно, что его величина γ_G будет постоянной для исследуемой транзисторной структуры. Для других архитектур коэффициент будет принимать другие значения. Тогда окончательное выражение для модели Пельгорма ЗМЗ кремневого конического GAA нанотранзистора сводится к простому выражению:

$$\sigma U_{Th} = \gamma_G G_s / \sqrt{S_{eff}}, \qquad (3)$$

Тогда отношение $\gamma_G / \sqrt{S_{eff}}$ определяет чувствительность отдельной транзисторной структуры к флуктуации ЗМЗ.

На рис. 2 показаны результаты расчетов по

модели Пельгрома для зависимости σUTh(Seff -1/2) для различных значений Gs:

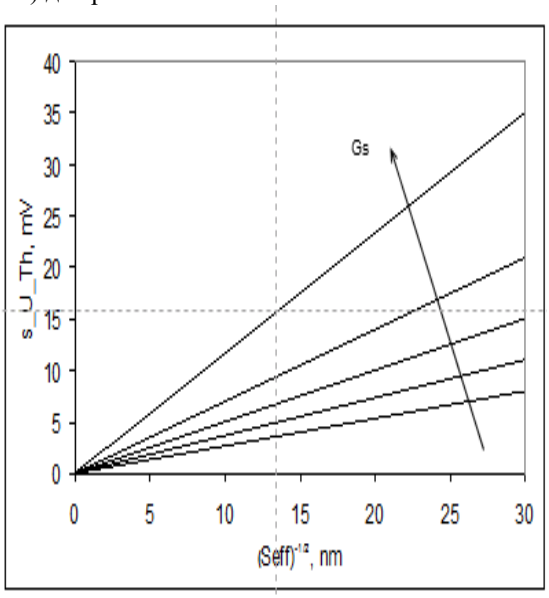

Рис. 2. Зависимость σUTh((Seff)-1/2) при Gs =(3; 4; 5; 8; 10) нм.

В данном случае в соответствии с классическими представлениями более тонкие структуры более чувствительны к проявлению механизма МЗМ [4, 5]. Значение параметра σUTh линейно увеличивается с ростом Gs и пропорционально уменьшается с ростом $\sqrt{S_{eff}}$.Следовательно, в приделе $\sqrt{S_{eff}} \rightarrow$ оо (для практических случаев при очень больших топологических размеров рабочей области) затвор будет вести себя как затвор с постоянным значением работы выхода φМ и параметр σUTh будет стремиться к нулю (σUTh→ 0). Наклон каждой зависимости σUTh((Seff)-1/2) определяет коэффициент Пельгорма ATh. Эта зависимость ATh (Gs) графически отражена на рис. 3 ниже.

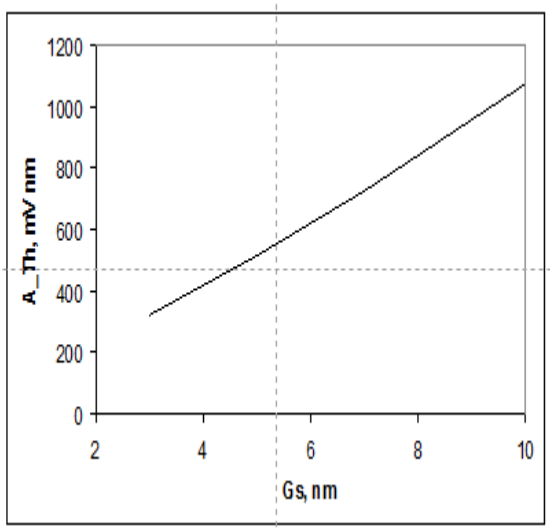

Рис. 3. Зависимость A_{Th}(Gs)

Наклон этой характеристики и есть значение параметра γ_G . В нашем случае коэффициент он равен 0.105 В. Такая точность необходима, для минимизации расхождения между прогноз и данными 3D моделирование параметра σUTh.
На рис. 4 приведены результаты 3D моделирования, полученные с использованием мульти метода 3D моделирования электро-физических характеристик кремниевых полевых с полностью охватывающим затвором нанотранзисторов [18].

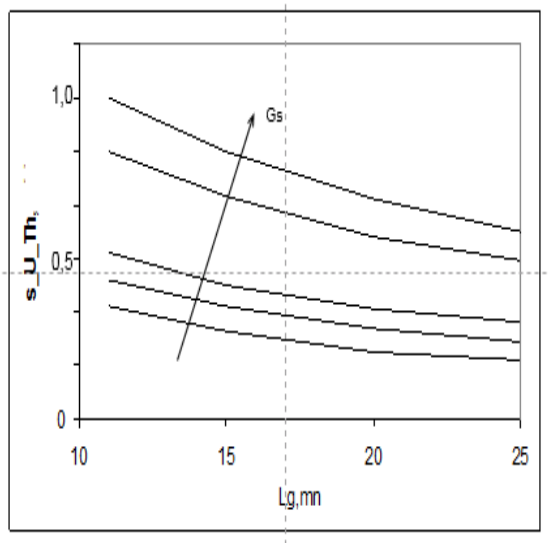

Рис. 4. Зависимость σUTh(Lg) при Gs =(3; 4; 5; 8; 10) нм, экстрагированная из результатов 3D моделирования

В исследуемом диапазоне топологических параметров транзисторов и размеров зерен в большинстве случаев подтверждается линейность зависимости σUTh((Seff)-1/2). При этом очевидно, что чем больше площадь затвора, тем меньше влияние металлических зерен. И, соответственно, наклон графиков σUTh((Seff)-1/2 для

разных значений Gs практические совпадает с коэффициентами Пельгрома и линейно увеличивается с увеличением Gs. Большие отклонения от линейности наблюдаются с чрезмерным ростом параметра σUTh в ультра тонких прототипах (с малыми значениями Dmin/Dmax и Lg) и большими зернами. Этот эффект возникает из-за того, что на затворе имеется всего несколько металлических зерен (например, для Lg=11 нм и Gs=10 нм их всего 4!) [19]. Поэтому, нормальное распределение порогового напряжения превращается в бимодальное с пиками вокруг экстремальных значений параметра φM (4,4/4,6 эВ) [12]. Это и обуславливает отклонение от линейности зависимости σUTh((Seff)-1/2 . Однако, можно выделить такой диапазон в отдельный случай, и откорректировать коэффициенты Пельгорма. Следует отметить, что более 95% рассматриваемого диапазона более или менее подчиняются линейному закону. Отклонение составляет менее 5%. Поэтому, полученные данные можно использовать для достоверного прогноза флуктуации порогового напряжения и связанными с ним других электро-физических характеристик транзистора.

Для среднего значения зернистости Gs=5нм представлен сравнительный анализ значений параметра σUTh полученные из модели основанной на теореме Пельгорма и 3D моделирования. На рис. 5 приведена зависимость σUTh(Seff) для прототипов в диапазоне длин Lg от 11 до 25 нм.

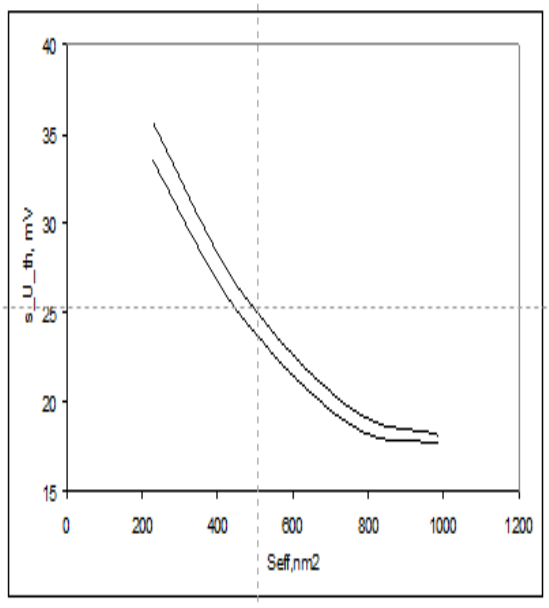

Рис. 5. Зависимость σUTh(Seff), где верхняя - данные по Пельгорму, нижняя по 3D моделированию.

Представленные данные говорят об прекрасном качественном и хорошем количественном совпадении. На рис. 6 и 7 приведены значения рассогласования параметра σUTh вычисленных разными способами и относительная погрешность между ними.

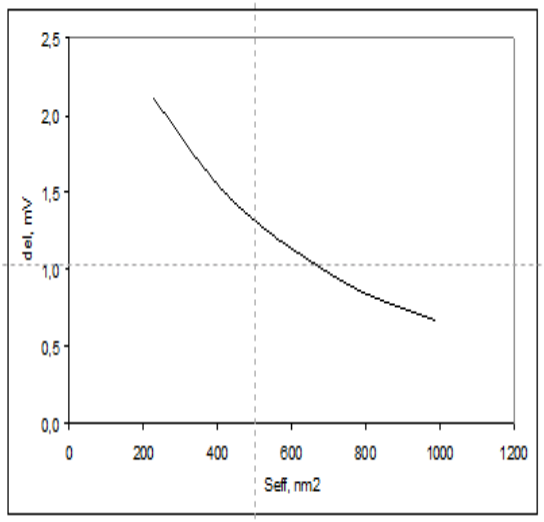

Рис. 6. Зависимость Δ(Seff)

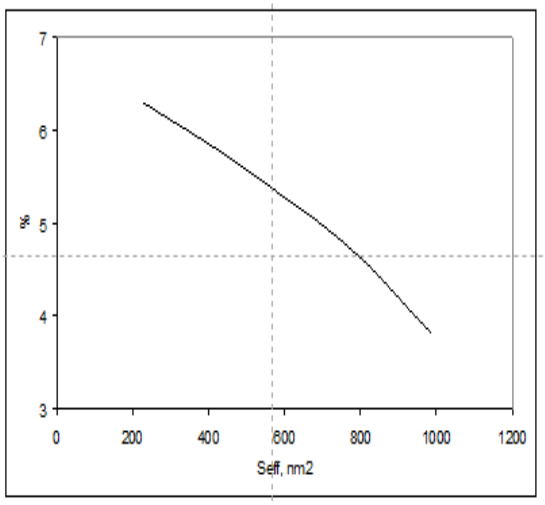

Рис. 7. Зависимость относительной погрешности (%) от Seff

Что и ожидалось, снижение длины затвора (и соответствующее уменьшение диаметров (процедура необходимая для подавления ККЭ)) приводит к росту погрешности. Чтобы избежать в дальнейшем дополнительного роста ошибок, например при расчете номиналов токов транзисторов, следует разбить весь диапазон топологических параметров на поддиапазоны и для них определить коэффициенты Пельгорма. При этом, важно подчеркнуть, для конкретной архитектуры транзистора с одинаковым металлическим затвором параметр γ_G не изменяется и определяется один раз! В упрощённом варианте для двух прототипов с топологическими параметрами соответствующие краям интересую-

щего диапазона моделируется флуктуация порогового напряжения из-за механизма ЗМЗ. Результаты моделирования для прототипов с длиной Lg 11 и 25 нм приведены ниже на рис. 8.

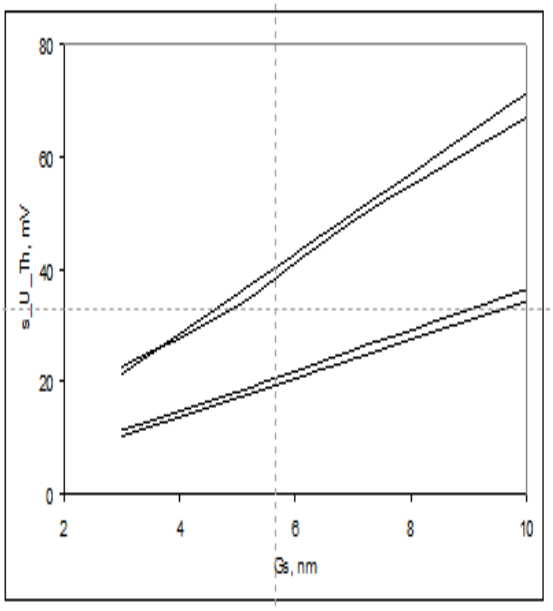

Рис. 8. Зависимость σUTh(Lg) при Lg = 11 нм (нижняя кривая в верхней группе) и 25 нм (нижняя кривая в нижней группе), верхние кривые в каждой группе аналогичные зависимости полученные при помощи (2) и (3)

Из 3D моделирования можно экстрагировать все значения параметров модели Пельгорма. И получить, что уже отмечалось, достоверную оценку флуктуации порогового напряжения.

4. Заключение

В диапазоне длин затворов транзисторов от11 до 25 нм и ЗМЗ от 3 до 10 ни получены коэффициенты Пельгорма. Относительные погрешности между модельными значениями σUTh данными полученными из 3D моделирования практически в 95% случаев ниже 5%.

Показано, что прототипы одинаковой архитектуры и с одним и тем же металлическим затвором, но имеющие разные топологические параметры, в частности размеры рабочей области, характеризуются одним и тем же значением параметра γ_G .

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН «Проведение фундаментальных научных исследований (47 ГП)» по теме № FNEF-2022-0022 "Математическое обеспечение и инструментальные средства для моделирования, проектирования и разработки элементов сложных технических систем, программных комплексов и телекоммуникационных сетей в различных проблемно-ориентированных областях".

The Influence of the Drain Size of the Metal Gate of Silicon Conical GAA Nanotransistors on the Fluctuations of the Threshold Voltage

N. Masalsky

Abstract. The influence of the grain size of a metal gate on the fluctuation of the threshold voltage of a silicon field GAA nanotransistor is discussed. Based on Pelgor's theorem, a method of reliable estimation of threshold voltage fluctuations has been developed. Pelgorm coefficients were obtained in the range of transistor gate lengths from 11 to 25 nm and average grain sizes from 3 to 10 nm. The relative errors between the model values of the standard deviation of the threshold voltage and the data obtained from 3D modeling are in almost 95% of cases below 5%.

Keywords: silicon all-around gate (GAA) nanotransistor, grain size of the metal gate, Pelgora coefficient, threshold voltage fluctuation, simulation

Литература

1. More Moore. International Roadmap for Devices and Systems. IRDS, Piscataway, NJ, USA, 2021

2. S. Yu, S. M. Won, H. W. Baac, D. Son, C. Shin. Quantitative evaluation of line-edge roughness in various FinFET structures: Bayesian neural network with automatic model selection. “IEEE Access”, vol. 10 (2022), 26340–26346

3. M. A. Elmessary. Scaling/LER study of Si GAA nanowire FET using 3D finite element Monte Carlo simulations. “Solid-State Electron”, vol. 128, (2017), 17–24.

4. K. Nayak, S. Agarwal, M. Bajaj, P. J. Oldiges, K. V. R. M. Murali, V. R. Rao. Metal-gate granularity-

induced threshold voltage variability and mismatch in Si gate-all-around nanowire n-MOSFETs. “IEEE Trans. Electron Devices”, vol. 61, (2014), no. 11, 3892–3895.

5. W.-L. Sung, Y.-S. Yang, Y. Li. Work-function fluctuation of gate-all-around silicon nanowire n-MOSFETs: A unified comparison between cuboid and Voronoi methods. “IEEE J. Electron Devices Soc.”, (2021), vol. 9, 151–159.

6. J. A. Croon, W. Sansen, H.E. Maes. Matching properties of deep sub-micron MOS transistors, Springer, 2005.

7. N.V. Masalskii. Simulation of silicon FETs with a fully enclosed gate with a high-k gate dielectric. “Russian Microelectronics’. (2023), V. 52, 228-232.

8. C. Akbar, Y. Li, W. L. Sung. Machine learning aided device simulation of work function fluctuation for multichannel gate-all-around silicon nanosheet MOSFETs. “IEEE Trans. Electron Devices”, (2021), vol. 68, no. 11, 5490–5497.

9. J. Lim, C. Shin. Machine learning (ML)-based model to characterize the line edge roughness (LER)-induced random variation in FinFET. “IEEE Access”, (2020), vol. 8, 158237–158242.

10. G. Indalecio, N. Seoane, K. Kalna, A. J. García-Loureiro. Fluctuation sensitivity map: A novel technique to characterise and predict device behaviour under metal grain work-function variability effects. “IEEE Trans. Electron Devices”, (2017), vol. 64, no. 4, 1695–1701.

11. H. Carrillo-Nuñez, N. Dimitrova, A. Asenov, V. Georgiev. Machine learning approach for predicting the effect of statistical variability in Si junctionless nanowire transistors. “IEEE Electron Device Lett.”, (2019), vol. 40, no. 9, 1366–1369.

12. M.J. Pelgrom. Matching properties of MOS transistors. “IEEE J. of solid-state circuits”, V. 24, (1989), 1433-1439.

13. S. A. Vitale, J. Kedzierski, P. Healey, P. W. Wyatt, C. L. Keast. Work-function-tuned TiN metal gate FDSOI transistors for subthreshold operation. “IEEE Trans. Electron Devices”, (2011), vol. 58, no. 2, 419–426.

14. K. Takeuchi, M.-S. Ibaraki, A. Nishida. Random fluctuations in scaled MOS devices. International Conference on Simulation of Semiconductor Processes and Devices SISPAD’09, SISPAD 2009, 79-85.

15. D. Nagy, G. Espiñeira, G. Indalecio, A. J. García-Loureiro, K. Kalna, N. Seoane. Benchmarking of FinFET, nanosheet, and nanowire FET architectures for future technology nodes. “IEEE Access”, (2020), vol. 8, 53196–53202.

16. Н.В. Масальский. Чувствительность распределения потенциала конических GAA нанотранзисторов к вариациям топологических размеров рабочей области, “Труды НИИСИ РАН”, 2023, Т. 13(3), 23-29.

17. J.P Colinge. FinFETs and Other Multi-Gate Transistor. NewYork: Springer-Verlag, 2008.

16. Н. В. Масальский. Проблемы моделирования 3D затворных полевых нанотранзисторов архитектура с полностью охватывающим затвором.”Нанотехнологии”. (2019), Т. 11, № 3, 14-24

19. J.S. Yoon, T. Rim, J. Kim, K. Kim, C.K. Baek, Y.H. Jeong. Statistical variability study of random dopant fluctuation on gate-all-around inversion-mode silicon nanowire field-effect transistors. “Appl. Phys. Lett.”, V. 106, (2015), 1035073.

Тенденции в графических ускорителях для высокопроизводительных вычислений

А. С. Шмелёв[1]

[1]МСЦ РАН филиал ФГУ ФНЦ НИИСИ РАН, Москва, Россия, guest8993@rambler.ru

Аннотация. Графические карты, построенные на основе большого количества простых и однотипных исполнительных устройств и обладающие высокой пиковой производительностью уже давно используются в области высокопроизводительных вычислений в качестве ускорителей вычислений. В настоящее время выпускаются отдельные продукты, ориентированные на применение в вычислительных центрах. В данной работе приводится обзор современных ускорителей для вычислительных центров, приведены их показатели производительности, а также приведены анонсы перспективных ускорителей вычислений и показаны тенденции в данной области.

Ключевые слова: высокопроизводительные вычисления, графические ускорители, ускорители вычислений, суперЭВМ, память с высокой пропускной способностью.

1. Введение

Все современные и ожидаемые в ближайшей перспективе процессоры общего назначения имеют архитектуру управления потоком команд. Порядок выполнения программ задаётся счётчиком команд, а для выдачи на выполнение нескольких команд в такт используется достаточно сложная и ресурсоёмкая аппаратура (системы выдачи команд не по порядку, позволяющие искать готовые к выполнению команды в окне до 200 команд, схемы предсказания переходов и т.д.), но и она не позволяет выдавать более 4-6 команд – такой темп выдачи держится уже много лет. Тактовая частота процессорных ядер тоже уже практически не увеличивается, и увеличение производительности процессоров достигается за счёт роста числа процессорных ядер и использования векторной обработки. Для построения высокопроизводительных вычислительных систем требуется использование очень большого количества процессорных ядер, что накладывает серьёзные ограничения на классы задач, которые могут эффективно использовать столь большое число процессоров и осложняет программирование. В настоящее время для построения суперкомпьютеров в дополнение к процессорам общего назначения широко применяются графические ускорители, которые на кодах с высоким уровнем однородного параллелизма позволяют достичь ускорения до нескольких десятков раз.

2. Графические ускорители

Основу практически всех графических ускорителей уже много лет составляют относительно простые однотипные исполнительные устройства, объединённые в группы с набором управляющей логики. У фирмы NVIDIA эти группы называются потоковыми мультипроцессорами Streaming Multiprocessor (SM). Все исполнительные устройства в пределах SM выполняют одну и ту же программу, причем в каждом ИУ может выполняться в режиме разделения времени несколько копий этой программы с разными данными. Современные ускорители поддерживают до нескольких тысяч нитей на SM, называемых тредами (threads). Переключение между тредами в каждом такте позволяет увеличить время между двумя последовательными командами в одном треде и компенсировать задержки вычисления результата в любом ИУ за исключением выборки данных из глобальной памяти, реализованной на динамической памяти. Как правило, один тред вычисляет один элемент в массиве результата, и все SM выполняют одну и ту же программу параллельной обработки данных. Последние поколения ускорителей вдобавок к массиву обычных ИУ имеют в своём составе и специальные ядра для работы с многомерными массивами (на практике матрицами) – тензорами. Такие ядра не столь универсальны в программировании, в сравнении с обычными, но на совмещённых операциях умножения и сложения (Fused Multiply Add) небольших матриц производительность этих ядер может значительно превосходить производительность всех классических ИУ ускорителя. Такие ядра используются как в области высокопроизводительных вычислений, так и в области машинного обучения.

Графические ускорители обладали огромным количеством процессорных ядер и огромной пиковой производительностью, начиная со времён их использования в качестве ускорителей вычислений [1,2], однако, на вычислительных задачах, максимальная реальная производительность графических ускорителей (GPU), как правило,

не превышает 50% от пиковой производительности. Одним из основных узких мест, затрудняющих повышение производительности связки процессор + ускоритель является недостаточная пропускная способность памяти. Из-за больших накладных расходов на пересылку данных между памятью центрального процессора (CPU) и памятью GPU для достижения максимальной производительности в GPU нужно чтобы обрабатываемый массив содержал тысячи элементов. А на малых длинах векторов, порядка пары сотен элементов, ускорения при выполнении программы на GPU вообще может не быть [3].

Для нивелирования этой проблемы в последних поколениях графических ускорителей, ориентированных на установку в вычислительных центрах, разработчики перешли с ранее традиционной для графических ускорителей памяти GDDR на память High Bandwidth Memory (HBM). В HBM с помощью сквозных кремниевых межсоединений (through-silicon vias (TSVs)) и микроконтактных выводов (microbumps) объединяются несколько кристаллов динамической памяти (включая опциональную базовую схему контроллера памяти) с дополнительным базовым кристаллом, который может включать в себя буферную схему и тестовую логику. За счёт этого удаётся обеспечить более высокую пропускную способность при меньшем расходе энергии и существенно меньших размерах по сравнению с классическими вариантами с DDR и GDDR [4], вместе с тем в HBM используется более широка шина данных, чем в GDDR. Так, в памяти стандарта HBM2E предусмотрено до 12 кристаллов памяти в вертикальном стеке общим объёмом до 24ГБ и шиной шириной 1024 бит с пропускной способностью 307ГБ/с на стек [5].

3. Обзор современных и перспективных ускорителей для HPC

На данный момент имеются три основных производителя, выпускающих ускорители для сферы высокопроизводительных вычислений, это компания Intel со своей линейкой продуктов Data Center GPU Max, NVIDIA с линейкой Data Center GPUs (до 2020 называлась Tesla), и AMD с линейкой Instinct.

Компания Intel выпустила в 2023 году ускоритель для вычислительных центров Data Center GPU Max 1550 на микроархитектуре с кодовым названием Ponte Vecchio [6], который содержит 128 X^e ядер. Каждое такое ядро содержит в себе 8 векторных исполнительных устройств, с длиной вектора 512 бит, и 8 матричных вычислителей, с размером матрицы 4096 бит, а также кэш первого уровня размером 512КБ и 8-и уровневый систолический массив. Производительность одного такого векторного арифметико-логического устройства (АЛУ) - 256 ФЛОП/такт над числами двойной точности (FP64), такая же производительность и в случае одинарной точности (FP32), либо удвоенная над числами половинной точности (FP16). Матричный же вычислитель не поддерживает операции над числами двойной точности, но может показать производительность 4096 операций в такт над числами половинной точности (FP16) или (BF16), 2048 операций в такт над TF32 и 8192 над INT8, т.е. матричный вычислитель более нацелен на задачи искусственного интеллекта и машинного обучения, нежели на высокопроизводительные вычисления. Шестнадцать таких ядер вместе с 16 устройствами трассировки лучей объединяются в один слайс (slice). Такие слайсы по 4 штуки вместе с кэш второго уровня и четырьмя контроллерами памяти HBM объединяются в стек и на кристалле находятся два таких стека [6,7]. Этот ускоритель имеет теоретическую производительность 52.43 ТФЛОПС на числах с двойной точностью при частоте 1600МГц (базовая частота 900МГц) и пропускную способность к 128ГБайтам памяти стандарта HBM2e в 3276.8 ГБ/с через шину шириной 8Кбит [8]. Также имеется возможность объединения до 8 GPU посредством высокоскоростных линий связи X^e Link и встроенного полного коммутатора. С процессором ускорители общаются через шину PCI-Express.

Более десяти тысяч вычислительных узлов из шести таких ускорителей, совместно с двумя процессорами серии Intel Xeon Max, установлены в суперкомпьютер Aurora, имеющий пиковую производительность более 2х ЭФЛОПС [9]. Надо отметить, что подобные процессоры имеют встроенную память HBM2e в размере 64ГБ, которая может использоваться и в качестве кэш памяти, и в качестве оперативной [10].

У компании AMD наиболее производительным ускорителем на данный момент является Instinct MI250X, выпущенный в ноябре 2021 года на архитектуре CDNA 2 [11]. В данном ускорителе объединены два кристалла по 110 вычислительных блоков каждый. В каждом таком вычислительном блоке имеется по 4 матричных вычислителя и 64 шейдерных ядра (4xSIMD16)[11], вместе с планировщиком задач и 16КБ кэш 1-ого уровня. На кристалле также имеется 8МБ кэш второго уровня (суммарно 16МБ на ускоритель). Все АЛУ поддерживают вычисления с двойной точностью, обеспечивая ускорителю производительность в 47.9 ТФЛОПС при пиковой частоте 1700МГц (базо-

вая частота 1000МГц). Такую же пиковую производительность InstinctMI250X имеет на числах с одинарной точностью (FP32), а на половинной FP16, bfloat16, а также INT8 пиковая производительность 383 ТФЛОПС. AMD также заявляет матричную производительность в 95.7 ТФЛОПС (FP64/FP32). Ускоритель имеет два блока по 64ГБ (один кристалл на архитектуре CDNA 2 может адресовать до 64ГБ) памяти HBM2e с 4КБитной шиной к каждому и суммарную пропускную способность в 3276.8 ГБ/с. Ускоритель имеет 8 каналов связи InfinityFabric (16бит на дуплексный канал, 50Гбит/с в каждом направлении) для связи с другими ускорителями/процессорами или шиной PCI-ExpressGen4 [11]. Наиболее эффективно эти линии связи могут использоваться в связки со специально адаптированными процессорами AMD EPYC, поддерживающими эту технологию, все остальные процессоры будут связываться по PCI-Express.

Подобные ускорители установлены во многих суперкомпьютерах, в частности в машинах HPE Cray EX235a, которые в разных вариациях, в зависимости от количества установленных вычислительных узлов, занимают верхние строчки в списках GREEN500, TOP500 и HPCG [12] (на данный момент установка Frontier из Ок-Риджской национальной лаборатории занимает первое место с производительностью на тесте Linpack 1,194.00 ПФЛОПС) [13].

В планах у AMD стоит выпуск ускорителей следующего поколения серии MI300. На момент написания статьи известно лишь, что это будет не просто ускоритель, а самодостаточный вычислительный узел, включающий в себя 24 универсальных ядра Zen 4 и неизвестное пока количество ядер графического ускорителя на архитектуре CDNA 3 с поддержкой памяти стандарта HBM3. Особенности архитектуры CDNA3 и количество ядер графического ускорителя компания AMD пока не раскрывает, известно лишь, что будет две версии ускорителя: Instinct MI300A с 24 ядрами Zen4, 128ГБ памяти HBM3 с общей памятью CPU+GPU и Instinct MI300X со 192ГБ памяти HBM3. Вычислительные узлы поколения MI300 уже устанавливаются в суперкомпьютер ElCapitan для Ливерморской национальной лаборатории, с ожидаемой производительностью свыше двух ЭФЛОПС. Запуск суперкомпьютера перенесли с 2023 на 2024 год, но до сих пор ожидается, что он займёт первую строчку в рейтинге TOP500 [14].

Компания NVIDIA стояла у истоков использования графических ускорителей в суперкомпьютерах и во многих машинах из первого десятка в списках TOP500 и GREEN500 используются ускорители этой фирмы разных поколений (в частности, суперкомпьютер Lenovo ThinkSystem SR670 V2 (Henri) с ускорителями H100 PCI-еанимает первое место в списке GREEN500) [15]. В 2022 году компания выпустила ускорители на микроархитектуре Hopper (H100), и в 22-23 годах на микроархитектуре с кодовым названием Ada Lovelace (L4 и L40).

Потоковый мультироцессор(SM) ускорителя H100 состоит из четырёх блоков, в каждом из которых имеется 16 АЛУ для арифметики над числами двойной точности, 32 для FP32, 16 для INT32, тензорному ядру четвёртого поколения, способному работать с форматами чисел FP8,FP16, BF16, TF32, FP64 и INT8, а также кэш команд уровня L0, варп-планировщик [16] и регистровый файл на 64КБ. Все четыре модуля имеют выход на раздельный кэш для команд и данных уровня L1 и модуль Tensor Memory Accelerator (TMA), для передачи больших блоков данных и многомерных тензоров между разделённой памятью и глобальной [17]. Пара таких мультипроцессоров объединяется в кластер обработки текстур (TPC). Теоретически, архитектура предусматривает до 72 TPC на кристалле, но существует два варианта ускорителей, с несколько разнящимися характеристиками. В варианте для PCIe 5-ого поколения с 57 TPC с пятью стеками памяти HBM2e общим объёмом 80ГБ и в варианте SXM5 с 66 TPC и пятью стеками памяти HBM3 такого же объёма. Примечательно, что только 2 TPC из всего массива способны работать с графикой, т.е. архитектура спроектирована именно для вычислительных центров. Оба ускорителя имеют по 50МБ кэш второго уровняи поддерживают каналыдуплексные каналы связи по 25ГБ/с в каждом направлении NVLink 4-ого поколения (600ГБ/с у версии PCI-e и 900ГБ/с у SMX5) и PCI-e пятого поколения. Для H100 SMX заявлена производительность 33,5 ТФЛОПС (FP64) и 66,9 ТФЛОПС (FP64 на тензорах) при пропускной способности к памяти в 3,35ТБ/с, а для H100 PCI-e 25,6 ТФЛОПС и 51,2 ТФЛОПС при 2ТБ/с соответственно [18].

Ускоритель для вычислительных центров L40 на самом большом кристалле (AD102) семейства Ada Lovelace имеет 18176 CUDA ядер против 16896 у H100 SMX. При этом производительность его на числах FP64 всего 1414 ГФЛОПС. Это объясняется разностью в организации потоковых мультипроцессоров – там всего по два АЛУ для FP64 на SM, и те, как заявляет сама NVIDIA, оставлены для поддержки кода с командами над FP64 [19]. Ускоритель имеет 576 тензорных ядер 4-ого поколения и 48Гбайт памяти стандарта GDDR6 с суммарной пропускной способностью 864 ГБ/с. Судя по более низ-

ким показателям производительности над числами с двойной точностью, ускорители этого семейства (речь о продуктах, предназначенных для вычислительных центров, а не о традиционных графических ускорителях GeForce) направлены более на графические приложения и задачи машинного обучения, чем на высокопроизводительные вычисления.

Таблица 1. Пиковая производительность графических ускорителей в ТФЛОПС

	NVidia H100 SXM5	**AMD MI250X**	**Intel X^e**
FP64	33.5	47.9	52.43
FP32	66.9	47.9	52.43
FP16	133.8	383	52.43
INT8	н/д	383	1664
Tensor FP64	66.9	95.7	Не поддерживает
Tensor FP32	494.7	95.7	419.4 (формат TF32)
Tensor FP16	989.4	383	838.9
Tensor INT8	1978.9	383	1677.7

Как видно из таблицы 1, нет однозначно более производительного ускорителя по всем параметрам. Также видно, что фирма NVIDIA утрачивает лидирующие позиции среди ускорителей для суперкомпьютерных центров, и простым увеличением пиковой производительности одного лишь ускорителя это теперь вряд ли будет легко исправить, т.к. Intel и AMD производят ещё и процессоры общего назначения, т.е. они могут применять, и применяют, в своих связках CPU+GPU дополнительные технологии, позволяющие повысить производительность вычислительного узла в целом (как то быстрые каналы связи, позволяющие повысить производительность связки из нескольких CPU и нескольких GPU). Пока ещё на верхних строчках рейтингов TOP500 и GREEN500 находится много установок с ускорителями NVIDIA, но если не будет предпринято ничего кардинального, эта ситуация изменится в худшую для NVIDIA сторону. Получается, что все три фирмы продвигают свою экосистему для вычислений, состоящую из средств программирования (CUDA, OpenMP/ROC и DC++/IntelOneAPI), ускорителей и процессоров общего назначения, только у фирмы NVIDIA нет своего процессора общего назначения, а конкурировать на два фронта будет значительно сложнее. Компания NVIDIA решила пойти по тому же пути, что и AMD, представив следующее поколение (с кодовым названием Grace Hopper) не просто ускорителей, но полноценных вычислительных узлов, сочетающих в себе 72 ядра процессоров общего назначения на архитектуре Arm Neoverse V2 и ядер своих графических ускорителей на уже известной архитектуре Hopper с поддержкой памяти стандартов HBM3 и HBM3e [20,21]. В этом вычислительном узле обещано уравнять пропускную способность GPU-GPU и CPU-GPU (по 900ГБ/с) для задач с большой загрузкой как GPU, так и CPU, также заявлена поддержка общего адресного пространства у CPU и GPU, что облегчит программирование, сократит количество пересылок данных и позволит повысить реальную производительность. Кроме того, заявлена поддержка ускорения кода на языках C++, Fortran и Python.

Как мы видим, имеется тенденция на увеличение пропускной способности к памяти – многие ускорители поддерживают память HBM (у Intel поддерживает также процессор Xeon Max), выпускаются специализированные модули (OAM,SXM) с компактным размещением ускорителей и памяти для сокращения линий связи, уже применяется память стандарта HBM3, объявлена поддержка памяти HBM3e, имеющей пропускную способность 8Гбит/с на контакт. В будущем использование памяти HBM-PIM, разработку которой анонсировала компания Samsung [22], позволит сократить количество обращений к памяти, за счёт организации простейших арифметических операций в самой памяти. Пока предполагается использовать данную технологию в задачах машинного обучения [23]. Идея наделить память возможностью самостоятельно выполнять простейшие арифметические действия не нова и предлагалась в своё время в ИППМ РАН [24] при разработке процессора с архитектурой управления потоком данных и, на самом деле, может найти применение и в ускорителях вычислений в машинах с традиционной архитектурой для снижения количества обращений в память, ускорения синхронизации и т.д., но пока о таких планах компаний-производителей мне не известно.

4. Заключение

Хочется отметить, что основной тенденцией развития ускорителей для суперкомпьютерных центров является повышение эффективности работы с памятью. Уже ушли от использования чисто графических ускорителей для построения суперкомпьютеров. Более того, на смену про-

стой связке CPU+GPU приходят специализированные вычислительные узлы, совмещающие в себе память с высокой пропускной способностью, универсальные вычислительные ядра и простые ядра ускорителей. И это происходит не столько ввиду технических ограничений, хотя имеются плюсы в близком расположении логики, организующей вычисления, основных вычислительных мощностей и памяти, сколько ввиду причин экономических – производители получают конкурентное преимущество, предлагая сразу законченную экосистему из программных и аппаратных решений.

Работа была выполнена в МСЦ РАН в рамках государственного задания по теме FNEF-2022-0016. В исследованиях использовался суперкомпьютер МВС-10П.

Graphics Accelerators for High-Performance Computing

A. S. Shmelev

Abstract. GPUs, built from a large number of relatively simple an similar execution units and having high peak performance, have been used long time in high-performance computing as accelerators. Nowday there are GPUs that were specially designed for use in supercomputer centers, and not as graphics processors.

Current trends in their development are presented.

Keywords: High performance computing (HPC), Graphics processing unit (GPU), Accelerated Processing Unit (APU), High Bandwidth Memory (HBM)

Литература

1. Lal Shimpi, Anand; Wilson, Derek (June 16, 2008). "Lots More Compute, a Leetle More Texturing - Nvidia's 1.4 Billion Transistor GPU: GT200 Arrives as the GeForce GTX 280 & 260". Anandtech.com. Retrieved December 11, 2015. https://www.anandtech.com/show/2549/3

2. Radeon R5xx Acceleration Revision 1.1 February 22, 2008 © 2008 Advanced Micro Devices, Inc.

3. D.B. Kirk, W.W. Hwu. Programming Massively Parallel Processors. A Hands-on Approach, Morgan Kaufmann Publishers, 2010, 258 p.

4. HBM: MemorySolutionforBandwidth-HungryProcessors, JoonyoungKimandYounsuKim, SKhynix // Hot Chips 26, August 2014

5. JEDEC Updates Groundbreaking High Bandwidth Memory (HBM) Standard (Пресс релиз). JEDEC. 2018-12-17.

6. Intel Data Center GPU Max Series Technical Overview https://www.intel.com/content/www/us/en/developer/articles/technical/intel-data-center-gpu-max-series-overview.html Обращение сентябрь 2023.

7. Cutress, Ian (August 24, 2021). "Hot Chips 2021 Live Blog: Graphics (Intel, AMD, Google, Xilinx)". AnandTech. Retrieved August 29, 2021.

8. Intel® Data Center GPU Max 1550 Specifications https://www.intel.com/content/www/us/en/products/sku/232873/intel-data-center-gpu-max-1550/specifications.html Обращение сентябрь 2023.

9. Argonne installs final components of Aurora supercomputer. Пресс релиз 22.06.2023 Jim Collins https://www.anl.gov/article/argonne-installs-final-components-of-aurora-supercomputer

10. Intel Xeon MAX 9480 Deep-Dive 64GB HBM2e Onboard Like a GPU or AI Accelerator. Patrick Kennedy 12.09.2023 https://www.servethehome.com/intel-xeon-max-9480-deep-dive-intel-has-64gb-hbm2e-onboard-like-a-gpu-or-ai-accelerator/3/

11. INTRODUCING AMD CDNA™ 2 ARCHITECTURE © 2021 Advanced Micro Device https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

12. https://www.top500.org/lists/ Обращение сентябрь 2023.

13. https://top500.org/lists/top500/2023/06/ Обращение сентябрь 2023.

14. https://www.anandtech.com/show/18946/el-capitan-installation-begins-first-apu-exascale-system-shaping-up-for-2024 Обращение сентябрь 2023

15. https://top500.org/lists/green500/2023/06/ Обращение сентябрь 2023.

16. Константин Калгин. Аппаратная архитектура CUDA. Новосибирский национальный иссле-

довательский государственный университет, Новосибирск, 2013. https://ssd.sscc.ru/sites/default/files/content/attach/332/cuda-3-arch.pdf

17. NVIDIA H100 Tensor Core GPU Architecture whitepaper V1.04 © 2023 NVIDIA Corporation. https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper

18. NVIDIA H100 Tensor Core GPU Datasheet © 2023 NVIDIA Corporation and affiliates. https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet

19. NVIDIA Ada Lovelace Professional GPU Architecture Whitepaper v1.1 © 2023 NVIDIA Corporation. https://images.nvidia.com/aem-dam/en-zz/Solutions/technologies/NVIDIA-ADA-GPU-PROVIZ-Architecture-Whitepaper_1.1.pdf

20. NVIDIA GH200 Grace Hopper Superchip Architecture Whitepaper V1.11 © 2023 NVIDIA Corporation. https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

21. NVIDIA Grace Hopper Superchip Data Sheet © 2023 NVIDIA Corporation. https://resources.nvidia.com/en-us-grace-cpu/grace-hopper-superchip

22. Samsung Develops Industry's First High Bandwidth Memory with AI Processing Power (Пресс Релиз 17.02.2021) https://news.samsung.com/global/samsung-develops-industrys-first-high-bandwidth-memory-with-ai-processing-power

23. Samsung Processing in Memory Technology at Hot Chips 2023 By Patrick Kennedy August 28, 2023 https://www.servethehome.com/samsung-processing-in-memory-technology-at-hot-chips-2023/

24. Яхонтов Д.Е., Левченко Н.Н., Окунев А.С. Принципы работы блока специальных операций модуля ассоциативной памяти параллельной потоковой вычислительной системы ППВС // Материалы Международной научно-технической конференции «Суперкомпьютерные технологии: разработка, программирование, применение» (СКТ-2010), Таганрог – Москва, 2010, Т. 1, С. 166-170.

От критических методов к процессам доказательств

В. Г. Редько[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, vgredko@gmail.com

Аннотация. Построена и проанализирована модель процесса формирования доказательства на основе использования критического мышления. Проведён анализ методов использования критических рассуждений в научном процессе. Проанализированная модель вносит вклад в изучение когнитивной эволюции.

Ключевые слова: критическое мышление, доказательство, научное познание

1. Введение

В настоящей работе строится и анализируется модель, иллюстрирующая возникновение доказательств на основе использования критического мышления. Будем отталкиваться от анализа, сделанного в книге В.Ф. Турчина «Феномен науки» [1]. В этой книге характеризуются процессы зарождения и начальные этапы развития научного познания. Отмечается важность возникновения критического мышления. Характеризуется переход от первобытного мышления к критическому (от мышления первобытных племен к возникновению формального логического мышления). Критическое мышление отличается от первобытного тем, что возникает оценка мыслительного процесса самим мыслящим субъектом [1, гл. 8]:

> «Критическое мышление рассматривает каждое объяснение (языковую модель действительности) наряду с другими, конкурирующими объяснениями (моделями), и оно не удовлетворится, пока не будет показано, чем данное объяснение лучше, чем конкурирующее».

Также в этой книге подробно рассматриваются связанные с критическим мышлением процессы возникновения математических доказательств, возникновения основанной на аксиомах Евклидовой геометрии в Древней Греции [1, гл. 10], в частности, отмечается следующее:

> «Ни в египетских, ни в вавилонских текстах мы не находим ничего, что хотя бы отдаленно было похоже на *математическое доказательство*. Понятие о доказательстве ввели греки, и это является их величайшей заслугой».

Отметим, что определённые формы критического мышления в Древней Греции были явно развиты, например, можно вспомнить беседы Сократа, который изначально мнимо соглашался с одним из участников беседы, а затем показывал ложность мнения оппонента. Смысл сократического метода состоял в исключении догм, осознании главного тезиса сократовской мудрости: «Я знаю, что ничего не знаю». Этот метод побуждает человека к самопознанию, способствует его самосовершенствованию. Метод Сократа способствовал устранению первоначальных мнений, появлению сомнений, тем самым расчищая дорогу для поиска истины. Некоторые диалоги Сократа представлены в трудах Платона [2].

Представляется, что критическое мышление способствовало и развитию математических доказательств. Но можно ли с помощью модели представить более детально переход от критического мышления к процессу доказательства. В настоящей работе строится достаточно простая модель, иллюстрирующая такой переход.

Будем рассматривать автономного агента, у которого уже имеется «чувство причинности». В работах [3, 4], отталкиваясь от анализа Дэвида Юма, который считал, что у нас имеется какое-то наше внутреннее чувство причинности, которое заставляет нас утверждать, что если за ситуацией A постоянно следует ситуация B, то A есть причина B [5], были построены и исследованы компьютерные модели автономных агентов с чувством причинности и без чувства причинности. Согласно моделям [3, 4] агенты с чувством причинности имеют преимущество по сравнению с агентами без чувства причинности, Более того, было показано, что в эволюционирующей популяции агентов в результате эволюции агенты с чувством причинности могут вытеснять из популяции агентов без чувства причинности [4].

В данной работе предполагается, что агенты имеют чувство причинности и могут запоминать элементарные связи между ситуациями во внешнем мире $S(t)$, своими действиями $A(t)$ и следующими ситуациями $S(t+1)$ [3]:

$$\{S(t), A(t) \rightarrow S(t+1)\}, \qquad (1)$$

где t – дискретное время, $t = 1, 2,\ldots$

2. Иллюстративная модель

Строим модель, предполагая следующее. Имеется агент, имеющий чувство причинности, т.е. агент может формировать и запоминать правила вида (1). Агент функционирует в достаточно простом лабиринте (рис. 1). Лабиринт состоит из клеток, светлым показаны клетки, в которых может находиться агент, черным показаны препятствия в лабиринте, т.е. агент не может попасть в эти клетки. Жирной линией показаны границы лабиринта, которые агент не может преодолеть, т.е. агент всё время находится в лабиринте.

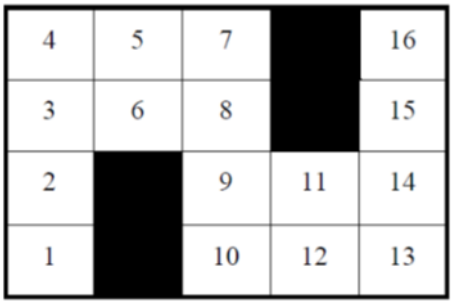

Рис. 1. Схема лабиринта, в котором находится агент.

Ситуация – клетка, в которой находится агент. Действие агента – переход агента из одной клетки в другую соседнюю клетку. Первоначально агент изучает лабиринт, запоминает клетки, в которых он находится, свои действия (переходы между клетками) и клетки, в которые он попадает в результате действий, т.е. агент непосредственно запоминает связи между ситуациями и своими действиями согласно выражению (1). Например, схема изучения лабиринта может быть следующая. Для каждой клетки лабиринта агент совершает все возможные переходы из этой клетки в соседние и запоминает все возможные связи вида (1). Считаем, что в результате этого изучения лабиринта агент полностью его осваивает, т.е. для всех возможных ситуаций (клеток) и действий (переходов в соседние клетки) он знает в какую ситуацию (клетку) он попадёт. При первоначальном изучении лабиринта пока ещё нет целевой клетки.

После изучения лабиринта агент выполняет определенный поиск. Сначала агент помещается в стартовую клетку 1 и ведет поиск цели. Цель находится в клетке 16, но агенту известно только то, что имеется цель, а место нахождения цели ему заранее неизвестно. Каждый такт времени агент перемещается на одну клетку. Агент не может выйти из лабиринта и не может попасть в клетку с препятствием. Также предполагается, что агент не может перемещаться в те клетки, в которых он уже был во время поиска. Из клетки 1 агент может переместиться только в клетку 2, затем – в клетку 3. Дальше имеются варианты: перемещаться в клетку 4 либо в клетку 6. При перемещениях он отслеживает свой путь в соответствии с выражением (1).

Если агент попал в клетку 8 из клетки 6, то в клетку 7 он переместиться не может, так как после этого он будет вынужден повторно зайти в одну из клеток 5, 6, 4, 3, а повторные входы в клетки ему не разрешены. Отметим, что такие запреты на повторные входы в клетки агент может обнаружить не сразу; считаем, что такие запреты агент обнаруживает методом проб и ошибок, например, он может пройти по клеткам 8->7->5->4 и обнаружить запрет (в клетку 3 повторно идти нельзя), считаем, что после обнаружения запрета, агент возвращается в последнюю клетку, из которой имеется два возможных перехода, т.е. в рассматриваемом случае агент возвращается в клетку 8. Следовательно, из клетки 3 агент должен будет пройти по одному из четырёх возможных ближних путей, а именно по следующим клеткам: 3->6->8 или 3->4->5->7->8 или 3->4->5->6->8 или 3->6->5->7->8. То есть, имеется четыре под-варианта прохода из клетки 3, во всех четырёх под-вариантах агент попадает в клетку 8.

Из клетки 8 агент перемещается в клетку 9. Далее, как и для клеток 3,4,5,6,7,8 у агента имеются четыре под-варианта дальнейшего пути: 9->11->14 или 9->10->12->13->14 или 9->10->12->11->14 или 9->11->12->13->14. Во всех этих под-вариантах агент попадает в клетку 14. Далее из клетки 14 он переходит в клетку 15, а затем в клетку 16, а этой клетке агент обнаруживает цель, после чего поиск прекращается.

Каждому из под-вариантов прохождения по клеткам 3,4,5,6,7,8 может быть сопоставлен любой под-вариант прохождения по клеткам 9, 10, 11, 12, 13, 14. Таким образом, существует 16 общих вариантов нахождения цели. Считаем, что после нахождения цели агент проверяет все эти 16 варианта прохождения к цели и критически оценивает их.

Агент критически оценивает варианты по числу проходимых в варианте клеток: чем больше проходится клеток в варианте, тем хуже этот вариант. И согласно правилу критического мышления, агент выбирает наилучший вариант, т.е. путь движения по клеткам:

1->2->3->6->8->9->11->14->15->16.

Формально путь к цели можно представить несколько подробнее, с учетом направления движения. Тогда наилучший вариант пути будет выглядеть следующим образом:

1 -> вверх -> 2 -> вверх -> 3 -> вправо -> 6 -> вправо -> 8 -> вниз -> 9 -> вправо -> 11

-> вправо -> 14 -> вверх -> 15 -> вверх -> 16.

Отметим, что поскольку после процесса первоначального изучения лабиринта агент полностью знает все элементарные переходы вида (1) для всех переходов между соседними клетками лабиринта, то поиск цели агент может проводить мысленно, без реального перемещения по клеткам. Нужно только, чтобы при мысленном попадании агента в клетку 16, агенту сообщалось, что цель находится в этой клетке. Более того, агент может сразу находить оптимальные части пути между клетками 3 и 8 (3->6->8) и между клетками 9 и 14 (9->11->14), не рассматривая более трудоёмкие пути между этими клетками.

Подчеркнём, что можно провести аналогию между рассмотренным агентом и процессом математического доказательства. В нашей модели мы можем считать, что переход агентом к новой ситуации (из одной клетки в соседнюю) мысленно соответствует одному элементарному логическому шагу и чем меньше таких логических шагов, тем доказательство того, что цель будет найдена, более эффективно. Например, можно рассмотреть доказательства положений (теорем) в «Началах» Эвклида [6, 7]. Каждое доказательство теоремы можно рассматривать как цепочку логических шагов, аналогично цепочке элементарных логических шагов агента в нашей модели. Образно говоря, в рассматриваемом случае агент доказывает такую «теорему».

> Если я нахожусь в стартовой клетке 1 изученного мной лабиринта, то существует путь к целевой клетке 16 этого лабиринта, причём кратчайший путь к целевой клетке таков: 1 -> вверх -> 2 -> вверх -> 3 -> вправо -> 6 -> вправо -> 8 -> вниз -> 9 -> вправо -> 11 -> вправо -> 14 -> вверх -> 15 -> вверх -> 16, где числа соответствуют номерам клетки, слова указывают направления перемещения между соседними клетками.

Таким образом, иллюстративная модель характеризует пример использования критического мышления при формировании доказательства.

3. Обсуждение

Естественно, что приведённая иллюстративная модель рассматривает простой пример как использования критического мышления, так и простой пример доказательства. Тем не менее, этот пример характеризует суть перехода от критического мышления к доказательствам.

Роль критического мышления, критического анализа в научном познании подчёркивалась рядом исследователей научного познания. В работах Карла Поппера процесс формирования новых теорий рассматривается в виде схемы [8]:

$$P_1 \text{ -> } TT \text{ -> } EE \text{ -> } P_2 \text{ ,}$$

где P_1 – исходная проблема, TT – пробное решение проблемы, пробная теория, которая может быть (частично или в целом) ошибочной, эта теория подвергается оценке, критическому устранению ошибок EE, в ходе оценки возникает следующая проблема P_2. Как подчёркивает Поппер, «оценка всегда является критической, её цель открытие и устранение ошибок».

Иллюстрируя роль критического метода, Поппер пишет:

> «Разница между амёбой и Эйнштейном состоит в том, что хотя оба используют метод проб и устранения ошибок, амёба не любит ошибок, а Эйнштейна они интересуют: он осознанно ищет у себя ошибки, надеясь узнать нечто благодаря их обнаружению и устранению. Метод науки – это критический метод».

Определённые формы критического мышления с детальным рассмотрением удачных и неудачных решений используются в современных системах искусственного интеллекта. Например, в работе [9] строится и исследуется модель автономных компьютерных агентов, которые могут самостоятельно планировать свои действия и оценивать качество этих действий. Суть схемы этой модели состоит в следующем. Агент пробует действие (или определённое решение вопроса), самостоятельно оценивает действие, и если действие оказалось ошибочным, то агент анализирует, в чём ошибка и как её можно исправить, после этого агент формирует новый план действий с учётом предыдущего опыта. Подробней см. работу [9]. Отметим, что в настоящее время в системах искусственного интеллекта активно используются обработка информации как из физического мира, так и из языкового мира. Например, в сократических моделях возможен диалог между внутренними модулями всей системы [10] с использованием информации из физического и языкового миров. В работе [9] также используется информация из физического мира и языкового мира.

Отметим возможные пути развития дальнейших моделей. Хотя рассмотренная модель упрощённая, отталкиваясь от этой модели и работ [1, 8, 11], можно анализировать более детально процессы математических выводов и процессы формирования новых теорий, новых научных дисциплин. Кроме того, интересно проанализировать, как развивались познавательные способности животных в процессе биологической эволюции [1, 11-13].

4. Заключение

Итак, в настоящей работе рассмотрена и проанализирована модель формирования доказательства на основе критического мышления. Конечно была использована довольно упрощённая схема критического мышления и упрощённая схема доказательства, тем не менее, эта изложенная модель характеризует переход от критического мышления к доказательствам. Эта модель – определённый вклад в анализ когнитивной эволюции [11], в результате которой произошло наше мышление, используемое в научном познании.

Настоящая работа выполнена в рамках государственного задания по проведению фундаментальных научных исследований по теме «Исследование нейроморфных систем обработки больших данных и технологии их изготовления», проект № FNEF-2022-0003.

From Critical Methods to Proof Processes

Vladimir G. Red'ko

Abstract. A model of the process of forming evidence based on the use of critical thinking was constructed and analyzed. An analysis of methods for using critical reasoning in the scientific process is carried out. The analyzed model contributes to the study of cognitive evolution.

Keywords: critical thinking, evidence, scientific cognition

Литература

1. В.Ф. Турчин. Феномен науки: Кибернетический подход к эволюции. М., Наука, 1993. (1-е изд.). М.: ЭТС, 2000. (2-е изд.). Turchin V.F. The Phenomenon of Science. A Cybernetic Approach to Human Evolution. New York: Columbia University Press, 1977. См. также http://www.refal.ru/turchin/phenomenon/

2. Платон. Сочинения в трех томах. (В четырех книгах). М., Мысль, 1968.

3. В.Г. Редько. Моделирование чувства причинности. Первые результаты. «Труды НИИСИ РАН», т. 9 (2019), № 1, 66–68.

4. В.Г. Редько. Модель чувства причинности. «Труды НИИСИ РАН», т.10 (2020), № 2, 34–38.

5. Д. Юм. Исследование о человеческом познании. Соч. в 2-х томах, т. 2. М., Мысль, 1966, 5–169.

6. Евклид. НАЧАЛА. Книги I–VI. (Пер. с греческого и комментарии Д.Д. Мордухай-Болтовского при редакционном участии М.Я. Выгодского и И.Н. Веселовского). М.-Л., ОГИЗ, Государственное издательство технико-теоретической литературы, 1948.

7. https://24smi.org/celebrity/4943-evklid.html

8. К.Р. Поппер. Объективное знание. Эволюционный подход. М., Эдиториал УРСС, 2002.

9. N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, S. Yao. Reflexion: Language agents with verbal reinforcement learning. arXiv preprint (2023): arXiv:2303.11366v4, https://doi.org/10.48550/arXiv.2303.11366

10. A. Zeng, M. Attarian, B. Ichter, K. Choromanski, A. Wong, S. Welker, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, P. Florence. Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language. arXiv preprint (2022): arXiv:2204.00598, https://doi.org/10.48550/arXiv.2204.00598

11. В.Г. Редько. Моделирование когнитивной эволюции: На пути к теории эволюционного происхождения мышления. Изд. 2, испр. и доп. М., URSS, 2018.

12. Л.Г. Воронин. Эволюция высшей нервной деятельности (очерки). М., Наука, 1977.

13. З.А. Зорина, И.И. Полетаева. Зоопсихология. Элементарное мышление животных. М., АспектПресс, 2001.

О нововведениях в цифровой образовательной платформе Мирера

И. А. Васильев[1], А. С. Караваева[2], А. Г. Леонов[3], К. А. Мащенко[4], А. В. Шляхов[5]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, vanya71161@gmail.com;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, aleksandrakaravaevaa@yandex.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, МГУ им. М. В. Ломоносова, Москва, Россия, МПГУ, Москва, Россия, Государственный университет управления, Москва, Россия, dr.l@vip.niisi.ru;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, kirill.mashchenko@vip.niisi.ru;

[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, shlyakhov@vip.niisi.ru

Аннотация. При опытной эксплуатации образовательных платформ могут возникать трудности и потребности в нововведениях или исправлениях, которые не были заметны во время разработки и тестирования. Своевременные доработки и изменения способствуют улучшению опыта использования и уменьшают количество сил, затраченных на выполнение неудобных действий в системе, что позволяет лучше сконцентрироваться на выполнении основных задач, таких как преподавание или обучение. В данной статье описываются проблемы, с которыми столкнулись пользователи во время использования комплекта модулей цифровой образовательной платформы Мирера, и методы их решения.

Ключевые слова: цифровая образовательная платформа, цифровая образовательная среда, Мирера, автоматическая проверка, машинное обучение, нейронные сети, трансформеры, семантическая проверка, WebSocket, модульная архитектура

1. Введение

В 2023 году был произведен ввод в опытную эксплуатацию масштабируемой отечественной интегрирующей цифровой образовательной платформы Мирера-2023 [1] для комбинированного очного и дистанционного обучения с автоматической проверкой успешности выполнения отдельных заданий и модулей и курса в целом, оптимизированная на российскую систему организации высшего и школьного образования [2].

По итогам опытной эксплуатации платформы выяснилась целесообразность доработки модулей, обеспечивающих:

1. Сбор и отображение статистики результатов студентов, а также всестороннюю коммуникацию в режиме онлайн преподавателя со студентами;
2. Семантическую проверку и проверку на плагиат решения студента;
3. Авторизацию пользователей и рассылку уведомлений через социальные сети;
4. Модульную архитектуру курсов и использование одинаковых модулей в разных курсах.

Также по итогам опытной эксплуатации обнаружилась необходимость добавления в Мирерy-2023 новой функциональности, а именно:

1. Новой архитектуры сбора и логики учета посещаемости студентов при гибридных занятиях;
2. Новой архитектуры хранения и логики редактирования расписаний по всем группам;
3. Новой архитектуры хранения и отображения курсов в учебных пространствах университетов.

Все эти доработанные и вновь разработанные модули войдут в комплект Мирера-2024.

Кроме того, в 2023 году были разработаны несколько новых наполнений платформы Мирера. В частности, два наполнения, посвященные математическим темам (см. ниже раздел 4). Оказалось, что введение новых модулей не потребовало никаких доработок платформы.

2. Модули, требующие доработки

2.1. Сбор и отображение статистики результатов студентов, а также всесторонняя коммуникация в режиме онлайн преподавателя со студентами

В комплекте модулей Мирера-2023 модуль, отвечающий за сбор и отображение статистики результатов студентов, был реализован с помощью внутреннего обновления страницы раз в фиксированный промежуток времени. Данный

подход обеспечивает гарантированно постоянно правильную статистику по группе, независимо от таких факторов, как изменение наполнения контеста, редактирование баллов или статуса решений самим преподавателем или других преподавателей, редактирования вариантов студентов, списка студентов группы и многих других. Однако такой подход при большом количестве студентов и нагрузки на систему имеет и ряд минусов. Раз в фиксированный промежуток времени приходится запрашивать заново всю статистику по студентам, что является большим и сложным запросом, нагружающим как сервера Мирепы, так и компьютер преподавателя, если новые данные сильно отличаются от старых. Помимо этого, при изменении каких-либо данных преподавателем они или появляются спустя время, или вызывается ручное обновление страницы, что отрицательно влияет на опыт использования платформы. Также в Мирепе-2023 статистика могла отображаться только по одной группе, что неудобно при параллельных общих парах у нескольких групп, а также при необходимости сравнить итоговые результаты по курсу среди всех групп.

В связи со всеми этими проблемами принято решение перевести статистику на технологию WebSocket [3], а также реализовать возможность просмотра статистики сразу по нескольким группам. WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером, используя постоянное соединение (рис. 1). Особенностью данного протокола является то, что сервер не только может отвечать на запросы из браузера, но и быть инициатором отправки сообщения.

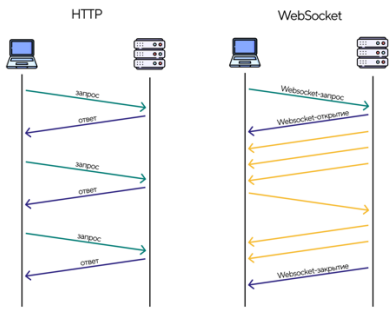

Рис. 1. Схема работы WebSocket

В комплекте модулей Мирепа-2023 уже использовалась технология WebSocket для обеспечения непрерывной коммуникации в режиме онлайн преподавателей и студентов. С ее помощью преподаватель может видеть процесс решения задания студентом, то есть процесс написания им каждого символа с точностью до положения каретки в файле программы. В обратную сторону, преподаватель может участвовать в процессе решения задания студентом, редактируя его решение. При этом у студента окошко решения блокируется, и он также полностью может наблюдать процесс написания решения преподавателем. Помимо этого, технология WebSocket используется в Мирепе-2023 для отображения в реальном времени результатов проверки задания тестирующим сервисом, а также для мгновенного отображения изменения преподавателем статусом попытки, баллов за задание, и многого другого.

Перевод всей статистики на технологию WebSocket решает сразу несколько проблем. С использованием этой технологии пропадает необходимость постоянного внутреннего обновления страницы раз в фиксированный промежуток времени: данные достаточно загрузить один раз при открытии страницы, после чего с помощью WebSocket они будут точечно обновляться лишь при необходимости по конкретному решению или студенту. Помимо этого, действия преподавателя теперь отображаются мгновенно, без обновления страницы и потери пользовательского удобства при использовании.

Однако использование этой технологии приводит и ко множеству проблем. Приходится отслеживать не только сокеты о действиях студентов, но и сокеты о действиях других преподавателей как в статистике, так и на любых других страницах, на которых можно изменять контент или данные курса или группы, ведь все эти действия также могут приводить к изменениям наполнения страницы. Помимо этого, при уходе со страницы статистики или приложения в целом при высокой загрузке компьютера соединение с сервером может быть разорвано и сокеты перестанут доходить до страницы сайта Мирепы, что приводит к потере данных и неактуальной картине для пользователя, что исправляется лишь обновлением страницы. Решением данной проблемы может быть возвращение к обновлению страницы раз в фиксированный, но уже увеличенный промежуток времени, или попытка узнать, когда соединение было разорвано и при его восстановлении или предлагать пользователю обновить страницу самому, или загружать данные без его ведома.

2.2. Семантическая проверка и проверка на плагиат решения студента

Комплект модулей платформы Мирепа-2023 позволяет проверять любые задания по программированию, написанных на множестве различных языков. Для проверки заданий используются метод прохождения студенческого решения

различных тестов с заданными входными и выходными данными. Входные данные подаются программе студента, после чего решение запускается на них, и полученные выходные данные сравниваются с эталонными, заданными преподавателем. Для быстрой генерации множества различных тестов в Мирере-2023 реализован модуль генерации входных и выходных данных тестов. Входные данные можно сгенерировать отдельным кодом, который генерирует случайные, а также «специальные» данные, что может быть полезно в задачах на базы данных, или, например, кодировку/декодировку сообщений. Выходные данные генерируются с использованием эталонного решения, которому на вход подаются сгенерированные или созданные преподавателем вручную входные данные [4].

Но даже при возможности быстрого создания большого количества случайных тестов, исключительно на основе тестов нельзя с уверенностью сказать, что студент использовал нужный метод при решении задания. В частности, от этой проблемы могут помочь фиксированные шаблонные строки, задаваемые преподавателем, которые студент не может удалить из своей программы и которые в любом случае будут выполняться вместе с решением студента, однако это является лишь частным решением проблемы. Полностью проверить правильность алгоритма студента предлагается с помощью системы антиплагиата, примененную к решению студента и эталонному решению преподавателя.

В Миреру-2023 встроена авторская система антиплагиата с возможностью обнаружения цепочек списывания между студентами и подсветкой списанных участков каждой пары решений [5]. Модуль антиплагиата Мирery-2023 исключает из проверки предоставляемый преподавателем для помощи студентам шаблон – набор готовых и защищенных от удаления и изменения участков решения, а также учитывает созданное преподавателем эталонное решение, игнорируя факты воспроизводства студентом его участков, так как это лишь означает, что студент хорошо усвоил материал. Благодаря этому проверяются только оригинальные фрагменты студенческого решения, что обеспечивает минимальное количество ложноположительных результатов.

Этот модуль можно применить к решению студента и решению преподавателя. Высокий процент плагиата на эталонное решение гарантирует, что студент использовал правильный алгоритм решения задания, игнорируя стилистические особенности написания кода студентом. Помимо этого, подсветка совпадающих строчек решения студента и эталонного решения студента помогают преподавателю быстро диагностировать ошибку в коде студента, поскольку совпадающие куски кода преподаватель сможет не рассматривать вовсе (рис. 2).

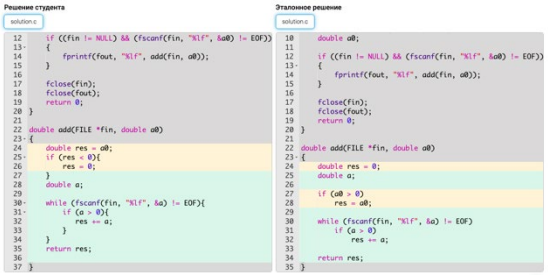

Рис. 2. Пример выделения совпадений кода студента с эталонным решением

2.3. Авторизация пользователей и рассылка уведомлений через социальные сети

В комплекте модулей Мирера-2023 используется единый модуль авторизации, также общий для цифровых образовательных сред ПиктоМир, ПиктоМир-К, КуМир, ЭВМ-практикум, разработанных в ФГУ ФНЦ НИИСИ РАН. Использование общего модуля авторизации позволяет обеспечить единую систему интеграции и сбора данных и решений студентов из всех цифровых образовательных сред на одной цифровой образовательной платформе [6].

В процессе опытной эксплуатации платформы Мирера-2023 обнаружились множество проблем при использовании единого модуля авторизации, из-за которых ее использование не было абсолютно бесшовным. Пользователям приходилось множество раз повторно авторизовываться в модуле авторизации даже для таких простых действий, как смена ФИО или пароля. В связи с этим сессии пользователя в Мирере-2023 и модуле авторизации были значительно интегрированы друг с другом, что обеспечило абсолютную бесшовность при использовании платформы и модуля авторизации одновременно.

В Мирере-2023 разработан модуль интеграции с ботами в социальных сетях ВКонтакте и Telegram. С его помощью студентам рассылаются уведомления о начале контеста с материалами для подготовки к занятию, напоминания о необходимости пройти контест и их статистикой по контестам. Преподавателю рассылаются уведомления о новых заявках студентов в группу, которые сразу же можно принять или отклонить (рис. 3), новых решениях студентов, требующих ручной проверки, статистика результатов студентов и многое другое.

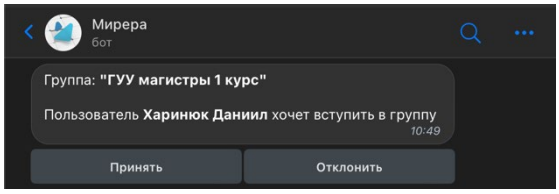

Рис. 3. Пример сообщения от бота преподавателю

Данный модуль был значительно доработан, добавлено множество новых уведомлений, а также обеспечена интеграция с социальной сетью ВКонтакте без необходимости регистрировать и настраивать новые группы в рамках социальной сети под каждую новую группу на платформе, теперь используется один общий бот для всех пользователей платформы. Помимо этого, добавлена возможность делать рассылки по всем студентам группы непосредственно со страницы группы или курса на платформе Мирера (рис. 4).

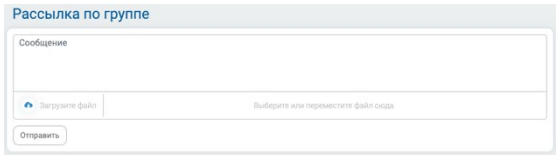

Рис. 4. Пример рассылки от бота из Мирры

2.4. Модульная архитектура курсов и использование одинаковых модулей в разных курсах

В 2023 году был разработан и апробирован в одном из московских университетов очно-дистанционный годовой вводный курс программирования для педагогических и технических университетов в рамках цифровой образовательной платформы Мирера.

Цифровая трансформация образовательных процессов подразумевает не только создание цифровых образовательных курсов или формирование гипертекстовых учебных материалов [7]. В Мирере-2023 разработана модульная архитектура организации курсов. Курс состоит из промежуточных курсов, которые в свою очередь состоят из тем, которые состоят из контестов. Данная структура позволяет не только организовать хранение больших, глубоких и сложных курсов, но и быстро переносить целые модули между различными похожими курсами, имеющими похожие темы. Также такая система позволяет быстро создавать курсы различными преподавателями, каждый из которых предоставляет свой модуль курса, из которых он в итоге соединяется в один.

Однако очень редко бывает ситуация, что модуль используется от курса к курсу в абсолютно одинаковом виде, часто приходится вносить в каждую его версию различные доработки и изменения. В этот момент появляется проблема наследования модулей курса, их версионирования, чтобы внесенные правки одновременно и применялись ко всем наследникам модуля, однако не затирали необходимые изменения в наследниках.

В рамках разработки и апробации курса, использующего модульную архитектуру хранения, были разработаны несколько вводных курсов программирования для педагогических университетов и технических университетов, имеющих ряд общих модулей, в том числе «Индуктивные функции», «Массивы», «Классы», «Деревья» и др. (рис. 5). Опытная эксплуатация этих курсов в ЦОП Мирера-2023 показала техническую осуществимость решений по реализации модульной структуры курсов в ЦОП Мирера-2023, а также подтвердила методическую эффективность создания общих модулей в курсах для разных категорий обучаемых.

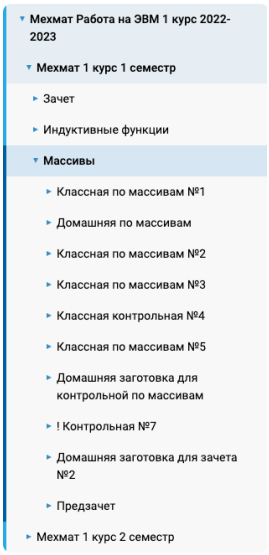

Рис. 5. Модульная организация курсов

Предоставляемые ЦОП Мирера-2023 возможности по сбору статистической информации о прохождении не только курса в целом, но и его отдельных модулей, по завершении осеннего семестра будут использованы для методической доработки общих модулей и содержащих эти модули вводных курсов программирования.

3. Модули, требующие добавления новой функциональности

3.1. Новая архитектура сбора и логики учета посещаемости студентов при гибридных занятиях

Поскольку апробация разработанного вводного курса по программированию проводилась как в очном, так и в дистанционном формате обучения, в Мирере-2023 встроен ряд возможностей по учету посещаемости и активности студентов во время занятий при всех вариантах формата обучения [8]. В случае дистанционного обучения подсистема проведения видеоконференций платформы Мирера регистрирует и запоминает всю информацию об активности студентов во время конференции, а также фиксирует время просмотра студентом записи пропущенной им видеоконференции.

В случае же очного обучения подобные инструменты становятся недоступны. Поэтому в Мирере внедрена возможность контроля посещаемости студентов по геопозиции: когда преподаватель приходит в аудиторию, он отправляет в платформу собственную геопозицию, после чего то же самое делают все студенты. Для тех редких случаев, когда телефон у студента по той или иной причине отсутствует, предусматривается возможность обращения студента к преподавателю по окончании занятия. Для финального контроля того, что не распознались лишние студенты, используется нейронная сеть по определению количества людей на фотографии, которой не нужны ни большое количество фотографий каждого студента, ни хорошие условия для фотографии, поскольку она лишь считает количество людей, а не распознает их личности [9].

Однако в процессе опытной эксплуатации выяснилось, что архитектура сбора и логика учета посещаемости студентов плохо подходит для гибридных занятий. Баллы за активность во время видеоконференции являются дополнительными и не связаны с баллами за посещаемость в целом. В итоге, хоть и получалось привести эти баллы к единому суммарному виду, неправильно считался сам процент посещаемости, который можно получить только при отправке геопозиции. Также добавление нового варианта учета посещаемости, например авторизация по QR-коду, не вписывается в архитектуру и в перспективе занимает много времени.

Поэтому архитектура перерабатывается под возможность использования сразу нескольких вариантов учета посещаемости, баллы за все эти варианты настраиваются в одном месте и являются баллами за посещение, которые можно набрать различными способами в зависимости от типа учета посещаемости. Каждый из этих типов можно настроить таким образом, чтобы выполнение его условий засчитывалось как посещение, или только выставлялись баллы студенту. Это полезно, например, когда преподаватель хочет настроить контест семинара так, чтобы посещение ставилось только тем студентам, которые присутствуют очно, а у студентов, которые присутствуют дистанционно или только просмотрели запись, учитывалась их активность, но не проставлялась посещаемость.

3.2. Новая архитектура хранения и логики редактирования расписаний по всем группам.

В комплекте модулей платформы Мирера-2023 встроена возможность настраивания расписания занятий курса отдельно и независимо по каждой группе, что необходимо, когда один курс проходит сразу несколько учебных групп. Время начала и конца контеста является относительным от начала темы, что позволяет быстро сдвигать расписание и перезапускать курсы. Однако в процессе опытной эксплуатации данной архитектуры выявился ряд проблем.

Выяснилось, что необходимо добавить какую-то логику повторяющихся событий с настройкой периода повтора, начала цепочки событий и так далее. Эта возможность необходима для быстрого создания, настройки и изменения повторяющихся занятий. Помимо этого, обнаружилась необходимость добавить возможность синхронизации расписаний между группами. Если просто синхронизировать их относительные расписания с разным началом темы - то этим не покроется случай, когда у групп расписание с разным интервалом между парами. Однако в рамках логики повторяющихся событий эта проблема может пропасть, если не синхронизировать время начала повтора. Также появилась необходимость возможности скрытия и настройки расписания не для всей группы, а для конкретных студентов, чтобы можно было открыть досдачу контеста конкретному студенту, или провести контест для одного студента.

3.3. Новая архитектура хранения и отображения курсов в учебных пространствах университетов.

В процессе опытной эксплуатации платформы Мирера-2023, при которой начали быстро появляться новые группы и курсы различных университетов и преподавателей, обнаружилась необходимость введения обобщающего понятия Пространства университета, чтобы студентам и преподавателям не отображались лишние группы и курсы, не относящиеся к их университету.

При таком подходе появляются несколько проблем. Один пользователь может находиться сразу в нескольких учебных пространствах, что поднимает вопрос администрирования учебных пространств. Помимо этого, пространство университета может быть открытым и закрытым, и, если оно закрыто, то также нужна новая роль администратора пространства университета. В таком случае требуется проработка, на основе каких данных и кто именно принимает заявки в пространство университета. Для этого может потребоваться необходимость сотрудничества непосредственно с учебной частью университета и последующей регистрацией пользователей, например, сразу на учебную почту. Также курсы могут быть привязаны как к какому-то пространству университета, так и быть общими для всех. Помимо этого, если один преподаватель преподает сразу в нескольких университетах – необходима возможность привязки курса сразу к нескольким пространствам университета.

Все указанные проблемы будут еще дополнительно изучаться в рамках разработки платформы Мирера-2024.

4. Примеры новых математических наполнений платформы Мирера

Для развития междисциплинарных навыков в Миреру были добавлены две проектные задачи: реализация класса Poly для работы с многочленами на языке Python, рассчитанная на учеников математических классов, и курс по выпуклой геометрии и коммутативной алгебре на языке Wolfram, рассчитанный на студентов младших курсов математических факультетов.

4.1. Быстрые алгоритмы умножения многочленов

Задача предполагает реализацию обучающимися задач арифметики многочленов. Арифметике многочленов посвящено значительное время курса Алгебры, поэтому к старшей школе обучающиеся очень хорошо знают наивные алгоритмы арифметических операций над многочленами. Так, сложение двух многочленов степени n “в столбик” требует $n + O(1)$ операций. Особый интерес представляет задача умножения двух многочленов. Ведь наивный алгоритм “в столбик” требует $O(n^2)$ умножений чисел, но простой алгоритм, основанный на быстром преобразовании Фурье, всего $O(n \log_2(n))$ [10]. Последний алгоритм укажем подробнее.

Пусть даны многочлены A, B, произведение C которых имеет степень 2n-1. Вычислим значения A и B на корнях из единицы степени 2n. Отсюда мы сразу получаем значения C в этих точках, по которым мы и восстановим C. Алгоритмы быстрого преобразования Фурье и обратного дискретного преобразования Фурье позволяют сделать это за $O(n \log_2(n))$ операций [11].

4.2. Исследование многочленов с помощью многогранников Ньютона

Курс состоит из нескольких разделов, посвященных изучению языка Wolfram через решение реальных математических задач с использованием как встроенных алгоритмов, так и реализованных самостоятельно. Задачи курса затрагивают следующие темы: многочлены Лорана, многогранники Ньютона, шеллинг выпуклого многогранника, идеалы, мономиальные базисы, базисы Грёбнера. Помимо этого, курс имеет ознакомительный раздел, призванный познакомить студентов с синтаксисом и базовыми методами языка Wolfram.

В рамках курса студентам предлагается реализовать описанный в статье [12] алгоритм нахождения мономиального базиса фактор-алгебры многочленов Лорана по идеалу, порождённому невырожденной системой многочленов с заданным симплициальным многогранником Ньютона. Это задание разбито на небольшие подзадачи, которые сгруппированы в смысловые разделы.

5. Заключение

В результате проведенной работы по разработке платформы Мирера-2023 и её апробации было выявлено преимущество использования технологии WebSocket на странице статистики над обновлением в фиксированный промежуток времени и необходимостью совмещения обоих способов. Было продемонстрировано применение модуля авторской системы антиплагиата для семантической проверки студенческих решений путем сравнения их с эталонным решением преподавателя. Глубокая интеграция общего модуля авторизации в платформу Мирера, а также использование социальных сетей для рассылок уведомлений улучшило опыт использования как для преподавателей, так и для учеников. Апробация курса, использующего новую модульную систему хранения, подтвердила методическую эффективность данного подхода.

В дальнейшем планируется доработка архитектуры модуля сбора посещаемости с возможностью учета посещаемости сразу несколькими способами для проведения гибридных занятий, а также возможность персонализации расписания для каждого студента и добавления периодичных событий для быстрого создания и редактирования расписаний курсов. Помимо этого,

планируется введение понятия пространства университета для более удобной фильтрации и отображения списков только относящихся к пользователю курсов.

Работа выполнена в рамках госзадания FNEF-2022-0010 «Разработка, реализация и внедрение семейства интегрированных многоязыковых сред программирования с автоматизированной проверкой заданий для учащихся образовательных организаций, ДОО, младшей, основной и старшей школы и студентов педагогических университетов».About Innovations in the Digital Educational Platform of Mirera

About Innovations in the Digital Educational Platform of Mirera

I. A. Vasilyev, A. S. Karavaeva, A. G. Leonov, K. A. Mashchenko, A. V. Shlyakhov

Abstract. During the trial operation of educational platforms, there may be difficulties and needs for innovations or fixes that were not noticeable during development and testing. Timely improvements and changes contribute to improving the user experience and reduce the amount of effort spent on performing inconvenient actions in the system, which allows you to better concentrate on completing basic tasks, such as teaching or studying. This article describes the problems that users have encountered while using the set of modules of the digital educational platform of Mirera, and methods for solving them.

Keywords: digital educational platform, digital educational environment, Mirera, automatic verification, machine learning, neural networks, transformers, semantic verification, WebSocket, modular architecture.

Литература

1. Платформа Мирера [Электронный ресурс] URL: https://www.mirera.ru (дата обращения: 11.10.2023)

2. Бахтеев О.Ю., Гафаров Ф.М., Гриншкун В.В., Дятлова О.В., Косарецкий С.Г., Кудинов В.А., Леонов А.Г., Сергеев А.Н., Щербатых С.В. Цифровая платформа образования, Вестник Российского фонда фундаментальных исследований, Том 1, No 113.

3. Шестаков В.С., Сагидуллин А.С. Применение технологии websocket в web-приложениях технологического назначения // Известия высших учебных заведений. Приборостроение. 2015. URL: https://cyberleninka.ru/article/n/primenenie-tehnologii-websocket-v-web-prilozheniyah-tehnologicheskogo-naznacheniya

4. Леонов А.Г., Дьяченко М.С., Мащенко К.А., Орловский А.Е., Райко И.Г., Райко М.В. Новые подходы к автоматизации проверки заданий в цифровых курсах, в сборнике ИНФОРМАТИЗАЦИЯ ОБРАЗОВАНИЯ И МЕТОДИКА ЭЛЕКТРОННОГО ОБУЧЕНИЯ: ЦИФРОВЫЕ ТЕХНОЛОГИИ В ОБРАЗОВАНИИ Материалы VI Международной научной конференции, место издания Красноярский государственный педагогический университет им. В.П. Астафьева Красноярск, тезисы, с. 173-179

5. Дьяченко М.С., Домрина В.А., Леонов А.Г., Мащенко К.А., Райко И.Г., Холькина А.А. Анализ методов проверки кода программ на плагиат в цифровой образовательной платформе Мирера, Труды НИИСИ РАН, Том 12, № 3.

6. Кушниренко А. Г., Леонов А. Г., Мащенко К. А., Орловский А. Е., Райко М. В., Шляхов А. В. Опыт интеграции цифровой образовательной среды КуМир в платформу Мирера, Статья в сборнике тезисов конференции «Объединённая конференция «СПО: от обучения до разработки»», с. 24-29

7. Леонов А.Г., Первин Ю.А. Качественные оценки эффективности методики обучения элементам информатики в пропедевтическом курсе // Ярославский педагогический вестник, №5, с. 92-96

8. Леонов А.Г., Мащенко К.А., Шляхов А.В., Холькина А.А. Подходы к учету посещаемости студентов в цифровой образовательной платформе Мирера, Труды НИИСИ РАН, Том 12, № 3

9. Иванова Е.В., Струева А.Ю. Система учета посещаемости на основе распознавания лиц // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2021 Т. 10, № 4 С. 60–73.

DOI: 10.14529/cmse210404.Seth Gilbert, Nancy Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, ACM SIGACT News, Volume 33, Issue 2, June 2002

10. А. Кушниренко, Быстрое преобразование Фурье, Научно-образовательный портал «Большая российская энциклопедия», Учредитель Правительство РФ, Номер 8, 2023, порядковый номер 9. DOI: 10.54972/00000001_2023_8_9

11. А. Кушниренко, Дискретное преобразование Фурье, Научно-образовательный портал «Большая российская энциклопедия», Учредитель Правительство РФ, Номер 8, 2023, порядковый номер 10. DOI: 10.54972/00000001_2023_8_10

12. A. Kushnirenko. Arnold's Piecewise Linear Filtrations, Analogues of Stanley–Reisner Rings and Simplicial Newton Polyhedra. // Mathematics 2022, 10 (23), 4445. DOI: 10.3390/math10234445. ISSN 2227-7390

Подписано в печать 15.11.2023 г.
Формат 60x90/8
Печать цифровая. Печатных листов 16.75
Тираж 100 экз. Заказ № 878

Отпечатано в ФГБУ «Издательство «Наука»
(Типография «Наука»)
121099, Москва, Шубинский пер., 6