Федеральное государственное учреждение «Федеральный научный центр
Научно-исследовательский институт системных исследований
Российской академии наук»
(ФГУ ФНЦ НИИСИ РАН)

ТРУДЫ НИИСИ РАН

ТОМ 12 № 4

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2022

Тематика номера:
Математическое моделирование и программное обеспечение технологических процессов, вопросы программирования, проектирование и моделирование СБИС, моделирование физических процессов в микро- и наноэлектронике

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, вопросы численного анализа, история науки и техники.

The topic of the issue:

Mathematical modeling and software of technological processes, programming issues, design and modeling of VLSI, modeling of physical processes in micro- and nanoelectronics

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, 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

I. МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ТЕХНОЛОГИЧЕСКИХ ПРОЦЕССОВ

С.Е. Базаева, Я.А. Зотов. Инструментальные средства разработки моделей технических комплексов 5

Т.К. Грингауз, А.Н. Онин. Пример импортозамещения программно-аппаратной архитектуры систем управления на вычислительных средствах ограниченной производительности 14

Я.А. Зотов. Обзор существующих решений в области моделирования технологических процессов 25

II. ВОПРОСЫ ПРОГРАММИРОВАНИЯ

В.А. Галатенко, К.А. Костюхин. Машинно-ориентированный текстовый интерфейс отладчика GDB 38

В.А. Галатенко, Г.Л. Левченкова, С.В. Самборский. Особенности сборки кросс-компилятора GCC и бинарных утилит 43

А.Б. Бетелин, А.А. Прилипко, Г.А. Прилипко, С.Г. Романюк, Д.В. Самборский. Логические ошибки в подсистемах ввода-вывода современных операционных систем 50

III. ПРОЕКТИРОВАНИЕ И МОДЕЛИРОВАНИЕ СБИС

П.О. Черняков, А.П. Скоробогатов. Использование при разработке СнК метода программной инжекции сбоев для оценки перехода на технологию с меньшими проектными нормами 56

Н.В. Желудков, К.А. Петров. Графовые нейронные сети и их применение при проектировании цифровых СБИС 61

А.Г. Ворсин, А.В. Шумаков, К.А. Петров, П.С. Зубковский. Использование метода косимуляции при разработке высокопроизводительных микропроцессоров 68

Я.М. Карандашев, Г.С. Теплов, В.В. Керемет, А.А. Карманов, А.В. Кузовков, М.Ю. Мальсагов. Исследование эффективности применения различных архитектур нейросетей в расчете маски в задаче инверсной фотолитографии 73

IV. МОДЕЛИРОВАНИЕ ФИЗИЧЕСКИХ ПРОЦЕССОВ В МИКРО- И НАНОЭЛЕКТРОНИКЕ

Н.В. Масальский. Теплопроводность тонких кремниевых эллиптических наноструктур 81

CONTENT

I. MATHEMATICAL MODELING AND SOFTWARE OF TECHNOLOGICAL PROCESSES

S. Bazaeva, Ya. Zotov. Tools for Developing Models of Technical Complexes 5

T.K. Gringauz, A.N. Onin. Control System Software and Hardware Architecture Import Substitution on Limited-Performance Computing Facilities.....14

Ya.. Zotov. Overview of existing solutions in the field of technological process modeling 25

II. PROGRAMMING ISSUES

V. Galatenko, K. Kostiukhin. Machine Oriented Text Interface to GDB.....38

V. Galatenko, G. Levchenkova, S. Samborskij. Building the GCC Compiler and Binary Utilities as Cross Tools 43

A.B. Betelin, A.A. Prilipko, G.A. Prilipko, S.G. Romanyuk, D.V. Samborskiy. Logical errors in the input/output subsystems of modern operating systems.....50

III. DESIGN AND MODELING OF VLSI

P. Chernyakov, A. Skorobogatov. Using Fault Injection Technique in SoC Design Flow to Evaluate Shifting to a Smaller Technology Node.....56

N.V. Zheludkov, K.A. Petrov. Graph neural networks and their application in the design of digital VLSI 61

A.G. Vorsin, A.V. Shumakov, K.A. Petrov, P.S. Zubkovskiy. Using the Cosimulation Method in High-Performance Microprocessors Development 68

Ya.M. Karandashev, G.S. Teplov, V.V. Keremet, A.A. Karmanov, A.V. Kuzovkov, M.Yu. Malsagov. Application of Various Neural Networks Architectures for Mask Calculation in Problem of Inverse Photolithography.....73

IV. MODELING OF PHYSICAL PROCESSES IN MICRO- AND NANOELECTRONICS

N. Masalsky. Thermal Conductivity of Thin Silicon Elliptical Nanostructures 81

Инструментальные средства разработки моделей технических комплексов

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 рассматриваются различные способы разработки моделей технических комплексов и анализируются достоинства и недостатки применяемых методов моделирования. На основе проведенного исследования авторами предлагается подход к созданию инструментальной среды моделирования, который позволяет конструировать стенды для широкого круга технологических процессов в различных областях применения, функционирующие в режиме реального времени. Инструментальные средства, используемые при моделировании, включают программное обеспечение и вычислительную аппаратуру.

Ключевые слова: технический комплекс, стенд полунатурного моделирования

1. Технический комплекс и его компоненты

Современный технический комплекс – это искусственно созданный объект, предназначенный для удовлетворения определенной потребности и существующий как процесс взаимодействия с компонентами окружающей среды, в результате которого производится желаемый результат. Технический комплекс (ТК) имеет сложную структурно-функциональную организацию и управляется с помощью автоматизированной системы управления. Элементами технического комплекса являются технологические процессы.

Технологический процесс – это часть производственного процесса, содержащая целенаправленные действия по изменению и/или определению состояния предмета труда [1]. В зависимости от области применения технического комплекса и свойств предмета труда технологический процесс (ТП) может быть промышленным, вычислительным, измерительным и т. п. Данные процессы функционально и логически связаны между собой, и за счет этого работа комплекса в целом направлена на достижение определенного результата. Как правило, технологические процессы в составе ТК образуют иерархию – более «сложный» ТП можно представить как совокупность последовательно или параллельно исполняемых более «простых».

Невозможно представить себе функционирование современного ТК без использования автоматизированной системы управления (АСУ). Автоматизированная система управления ТК – это совокупность аппаратных и программных средств, а также персонала, предназначенная для управления различными технологическими процессами в рамках технического комплекса. Каждый ТП с точки зрения АСУ является объектом управления.

Физически в составе ТК может быть задействовано различное оборудование, средства контроля и управления (в том числе, вычислительная техника и программное обеспечение), возможно, окружающая среда (например, при добыче полезных ископаемых), а также обслуживающий персонал.

Рис. 1. Пример структуры технического комплекса

Примерами ТК могут служить: комплекс по добыче и транспортировке нефти, атомная электростанция, система управления летательным аппаратом, интернет-магазин со складом товаров и службой доставки, система видеонаблюдения вокзала и привокзальной территории и т. п. Таким образом, современный технический комплекс – это многоуровневая динамическая система, состоящая из взаимодействующих разнородных сущностей.

В простейшем случае технический комплекс состоит из ЭВМ, управляющей подключенным физическим оборудованием (датчиками, исполнительными устройствами, индикаторами и пр.).

Более сложная конфигурация может включать управляющую ЭВМ (возможно, со SCADA-системой), дополнительную ЭВМ для обеспечения интерфейса с физическим оборудованием (например, программируемый логический контроллер, ПЛК) и внешние устройства (рис. 1). Программное обеспечение всех используемых ЭВМ также входит в состав технического комплекса.

Большие технические комплексы обычно включают несколько подсистем, каждая из которых реализует определенную функцию в составе

комплекса и имеет структуру, аналогичную показанной на рис.1.

2. Модель технического комплекса

Моделирование является неотъемлемой частью жизненного цикла современного технического комплекса, включая этапы проектирования, исследования, модификации и штатной эксплуатации.

Под моделью технического комплекса будем подразумевать такой объект, который способен замещать данный комплекс за счет обеспечения важных (с точки зрения решаемых задач) характеристик и свойств оригинала [2].

Классификация моделей, а также место и роль модели на всех этапах жизненного цикла объекта-оригинала определены государственными стандартами Российской Федерации, в том числе:

- 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].

Важнейшим качеством модели, обеспечивающим достоверность результатов моделирования, является адекватность поведения модели поведению объекта-оригинала (с учетом условий задачи, решаемой при моделировании).

Например, если программное обеспечение объекта-оригинала в штатном режиме должно работать с определенной скоростью (гарантировать время реакции на событие, соответствие реальному времени и т.п.), необходимо обеспечить соответствующие условия и в процессе моделирования.

При создании модели важно также учитывать экономический аспект – затраты не должны радикально увеличивать бюджет объекта-оригинала; напротив, модель должна способствовать экономии средств, особенно при использовании ее в качестве прообраза на этапе проектирования и на этапе испытаний.

Еще один существенный момент связан с процедурами верификации и валидации модели. Необходимо разрабатывать данные процедуры параллельно с созданием модели; в противном случае достоверность результатов моделирования будет вызывать сомнения.

3. Методы моделирования

Основными методами моделирования являются:

- физическое моделирование;
- математическое моделирование;
- полунатурное моделирование.

При физическом моделировании для каждого объекта-оригинала создается уникальная модель, например, модель самолета для исследования характеристик в аэродинамической трубе или полноразмерная модель кузова автомобиля из пластилина, чтобы визуально оценить дизайн. Изготовление сложных физических моделей требует значительных затрат (а в большинстве случаев создание такой модели невозможно), поэтому, несмотря на определенные преимущества, данный метод имеет очень ограниченную область применения.

Метод математического моделирования имеет большую историю, применяется очень широко и в настоящее время подразумевает использование ЭВМ [6]. Данный метод требует формализации поведения объекта-оригинала, например, представления в виде системы уравнений. Тогда, указав начальные условия, можно выполнить расчет и узнать, каким будет состояние объекта-оригинала в заданный момент времени. Однако современные технические комплексы настолько сложны по составу и функциональным связям, что с трудом поддаются формализации, поэтому на практике математическое моделирование чаще всего выполняется на уровне отдельного технологического процесса.

Математическая модель с высокой степенью детализации, требующая большого объема вычислений, обычно функционирует в масштабированном модельном времени, отличающемся от реального, что также необходимо учитывать при анализе результатов ее использования.

Потребность в моделировании технических комплексов со сложными структурными и функциональными связями привела к развитию отдельной ветви математического моделирования – методу имитационного моделирования.

В случае имитационного моделирования структура модели соответствует структуре компонентов (подсистем) объекта-оригинала. Способы формализации компонентов могут учитывать их особенности и быть различными для разных компонентов, например, вместо системы уравнений задействуют наборы данных, полученных экспериментальным путем в ходе других исследований или полученных для аналогичного объекта. Время может быть масштабированным.

Метод имитационного моделирования предполагает проведение множества экспериментов,

в результате которых получается статистическая оценка параметров моделируемого объекта. Функциональные связи внутри имитационной модели реализуют с помощью передачи данных между моделями подсистем. Более детальное отражение характеристик и свойств объекта-оригинала, достигаемое методом имитационного моделирования по сравнению с «чисто математическим» методом, повышает адекватность модели оригиналу.

Полунатурное моделирование – это следующий шаг по совершенствованию модели в плане соответствия объекту-оригиналу. Метод полунатурного моделирования предполагает частичное использование реальной аппаратуры в составе модели (моделирующего комплекса). Включение штатных компонентов объекта-оригинала в состав модели освобождает от необходимости детализации математической модели и одновременно обеспечивает более полное соответствие оригиналу – ведь согласно словам, которые приписывают Норберту Винеру, наиболее совершенной моделью кота является такой же кот, а лучше – он сам.

В случае полунатурного моделирования технического комплекса под реальной аппаратурой прежде всего подразумеваются штатные ЭВМ, входящие в состав технического комплекса и оснащенные штатным программным обеспечением, а также (при технической возможности) прочие электронные/электрические компоненты объекта-оригинала: интерфейсные устройства, периферийные электронные устройства, пульты, индикаторы и т.п.

Данный метод широко применяется, например, при создании тренажеров для обучения операторов: обучаемое лицо использует штатный пульт управления, а математическая модель в реальном масштабе времени обеспечивает функционирование пульта с учетом действий оператора.

Метод полунатурного моделирования с использованием штатной вычислительной аппаратуры позволяет также добиться функционирования программного обеспечения объекта-оригинала в штатном режиме, что повышает качество результатов моделирования.

Пример структуры моделирующего комплекса показан на рис. 2. Здесь имитационная полунатурная модель ТК повторяет структуру объекта-оригинала (рис. 1) – в модели представлена штатная вычислительная аппаратура (в технически допустимом объеме) и соответствующее программное обеспечение. Модель технического комплекса также включает ЭВМ, на которой работают математические модели компонентов объекта-оригинала, не представленных натурно.

Рис. 2. Структура полунатурной имитационной модели технического комплекса

Заметим, что некоторые элементы физического оборудования в модели, представленной на рис. 2, могут присутствовать натурно – к их числу относятся, например, световые индикаторы и другие аналогичные «несложные» устройства, если они являются просто выходными по отношению к ПЛК.

Другие устройства, которым для функционирования в составе модели все-таки необходима имитация входных и выходных данных, после технической доработки могут сохранить штатный интерфейс с ПЛК; тогда математическое моделирование для них будет происходить на более низком уровне.

Сохранение штатных интерфейсов в возможно полном объеме особенно важно при работе с объектами, функционирование которых требует планирования времени на уровне сотых долей секунды, поскольку математическое моделирование процессов на столь детальном уровне чрезвычайно затруднено или невозможно.

4. Стенд для полунатурного моделирования технического комплекса

Совокупность всех вычислительных средств, дополнительной аппаратуры и программного обеспечения, используемая при моделировании функционирования технического комплекса, образует стенд технического комплекса [7].

Стенд, на котором работает имитационная полунатурная модель технического комплекса, включает:

– штатное программное обеспечение и вычислительную аппаратуру объекта-оригинала (частично, в объеме, допустимом с технической точки зрения);
– математические модели компонентов объекта-оригинала, не представленных натурно, и ЭВМ, на которых исполняются эти математические модели;
– аппаратуру, имитирующую функционирование физических устройств объекта-оригинала (например, разрывные коробки на кабелях для передачи данных, генераторы сигналов (в том числе помех для беспроводной связи), пульты, индикаторы и т.п.); наличие и способы использования данной аппаратуры, а также соответствующего программного обеспечения зависят от решаемой задачи;
– инструментальную программно-аппаратную

среду для разработки, исследования и эксплуатации модели и ЭВМ, на которой функционирует данная среда;
- базу данных для результатов моделирования и ЭВМ, на которой функционирует база данных.

Логическая схема стенда для полунатурного моделирования работы технологического процесса показана на рис. 3. В состав стенда включены штатные ЭВМ объекта-оригинала с функционирующим на них штатным программным обеспечением. Эти ЭВМ связаны с базой данных, в которую помещается информация о функционировании штатных элементов стенда. Такая возможность накопления рабочих данных может присутствовать и в составе объекта-оригинала. Тогда накопление данных о функционировании модели и стенда (получаемых от инструментальной среды и средств имитации физического оборудования) следует производить либо в единой (штатной) базе данных, либо в другой базе данных, специально предназначенной для этих целей. Выбор варианта использования базы данных зависит от конкретного случая применения.

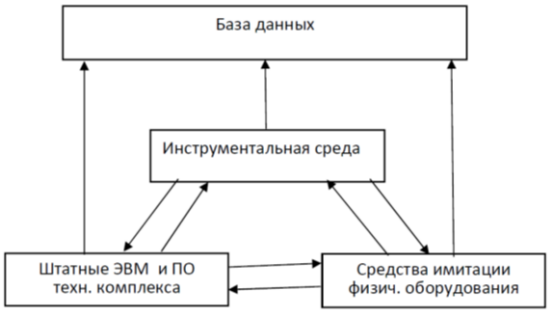

Рис. 3. Логическая схема стенда для моделирования работы технологического процесса

Средства имитации физического оборудования потенциально способны выполнять двунаправленный обмен данными со штатным оборудованием, включенным в состав стенда. Роль средств имитации физического оборудования состоит в том, чтобы «подыграть» штатному оборудованию и ПО.

Состав, назначение и алгоритм функционирования средств имитации зависит от конкретной решаемой задачи и технических возможностей.

Средства имитации физического оборудования включают математические модели устройств объекта-оригинала, не представленных на стенде натурно, а также штатные устройства, адаптированные для стенда, с (частично) сохраненным штатным интерфейсом и специальные устройства для управления процессом моделирования. Примерами адаптированных устройств могут служить световые индикаторы, экраны для визуализации, пульты управления, микропроцессорные блоки, подключенные к промышленному интернету вещей, и прочие подобные устройства, специальные устройства – это, например, разрывные коробки, имитирующие отказ при передаче данных, имитаторы помех беспроводной связи. Данные о функционировании средств имитации физического оборудования могут быть в случае необходимости запротоколированы и сохранены в базе данных.

Инструментальная программно-аппаратная среда предназначена для разработки, исследования и эксплуатации модели. Инструментальная среда может работать в двух режимах: в технологическом режиме и в режиме моделирования.

В технологическом режиме выполняется настройка параметров программ, имитирующих работу устройств, не представленных на стенде натурно, а также прочих аппаратно-программных параметров стенда: начальных условий, режимов передачи данных, протоколирования, отображения информации и т.п.

В режиме моделирования инструментальная среда служит для запуска/остановки модели, накопления данных и оперативного воздействия на ход моделирования, например, для имитации аварийных ситуаций.

5. Инструментальные среды для разработки моделей технических комплексов

Имитационное моделирование имеет большую историю и в настоящее время является одним из самых распространенных инструментов исследования сложных систем, поэтому на рынке программного обеспечения присутствует более сотни продуктов, реализующих методы имитационного моделирования [8].

Разработками первого поколения (1950-е гг.) были модели на языках высокого уровня FORTRAN и ALGOL без какой-либо специальной поддержки. Затем в 1960–1965 гг. появились языки моделирования GPSS (язык транзактов), SIMULA (язык процессов), SIMSCRIPT (язык событий) и прочие. Наконец, в настоящее время имитационное моделирование стало доступным не только высококвалифицированным программистам и разработчикам, но и более широкому кругу пользователей за счет автоматизации этапов построения модели и активного применения мультимедийных технологий, включая трехмерную графику и анимацию. Современные системы включают также возможности планирования экспериментов, управления моделированием, анализа результатов и поиска оптимальных решений, имеют средства технологической поддержки распределенного имитационного моделирования в сетях ЭВМ.

Среди широко применяемых программных продуктов можно выделить два основных варианта систем: специализированные компьютерные среды имитационного моделирования (например, AnyLogic, GPSS World) и стандартные математические системы, в которые включены средства имитационного моделирования (например, системы Matlab, Mathcad, Mathematica с пакетом Simulink). Также встречаются системы, ориентированные на отдельные виды имитационного моделирования (агентное, динамическое, дискретно-событийное моделирование) или на конкретные области применения (технологические процессы химического производства, конструирование в машиностроении и т.д.).

Развитые имитационные среды не требуют программирования в виде последовательности команд – пользователи формируют с помощью графического интерфейса модель из библиотечных модулей, и/или заполняют специальные формы. Во многих случаях среда разработки обеспечивает также возможность визуализации процесса имитации.

В отличие от систем имитационного моделирования, ориентированных на математические модели, среда полунатурного моделирования сама по себе не допускает широкого «универсального» применения, поскольку она должна включать определенный набор реальной аппаратуры для исследования конкретного объекта-образца.

Расширение области применения среды полунатурного моделирования происходит за счет обеспечения доступности большого набора аппаратных интерфейсов, охватывающего достаточно широкий круг задач. Для формирования библиотек математических моделей задействуют многофункциональные среды математического моделирования, например Matlab с пакетом Simulink. Примером такой среды является комплекс полунатурного моделирования КПМ Ритм [9].

По мере увеличения числа разработок, связанных с задачей имитационного моделирования (как для полунатурных, так и для чисто математических реализаций), возникла задача обеспечения повторного использования компонентов и их интероперабельности. Для решения этой проблемы предлагается технология HLA (High Level Architecture), оформленная в виде стандарта IEEE1516 [10, 11]. Данная технология представляет собой совокупность методик, соглашений и алгоритмов, которые позволяют задействовать при разработке новой среды созданные ранее имитационные модели и системы, а также обеспечивают возможность их функционирования на различных аппаратных конфигурациях. Технология HLA является стандартом, применяемым при решении задач имитационного моделирования в США.

6. Реализация стенда полунатурного моделирования

Рассмотрим более подробно процесс создания и функционирования стенда полунатурного моделирования на основе штатной вычислительной аппаратуры объекта-оригинала. Логическая схема стенда показана на рис. 3.

Данный стенд предполагается использовать для проведения исследований функционирования технического комплекса (рис. 1) методом имитационного полунатурного моделирования.

Состав реальной аппаратуры, включаемой в состав стенда, определяется техническими возможностями. Для примера используем базовый вариант, структура которого показана на рис. 4.

Стенд включает два штатных вычислителя – ЭВМ, на которой функционирует SCADA-система, управляющая работой технического комплекса, и ПЛК, который в штатном режиме обеспечивает интерфейс с периферийными устройствами объекта-оригинала. Предположим, что база данных, накапливающая информацию о функционировании объекта-оригинала и стенда, включена в состав управляющей ЭВМ. Вынесение этой базы на дополнительный компьютер не добавит существенных сложностей в работу стенда.

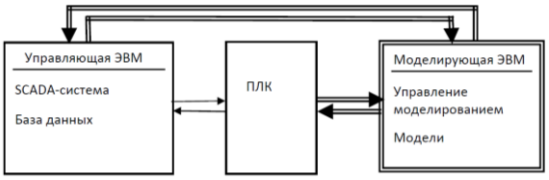

Рис. 4. Структура стенда полунатурного моделирования

При конструировании стенда полунатурного моделирования основная задача состоит в сохранении максимально возможного числа компонентов и характеристик объекта-оригинала. На рис. 4 сплошными линиями черного цвета показаны элементы и интерфейсы, присутствующие в штатном варианте, двойными линиями – элементы и интерфейсы, требуемые для функционирования стенда (нештатные).

Адекватность модели объекту-оригиналу является одним из ключевых условий успеха процесса моделирования, и с этой точки зрения стенд полунатурного моделирования дает следующие преимущества: во-первых, структура модели соответствует структуре компонентов объекта-оригинала, что облегчает построение и понимание процесса моделирования; во-вторых,

такая модель обеспечивает более детальное отражение характеристик и свойств объекта-оригинала за счет включения реальной аппаратуры, функционирующей в штатном режиме; в-третьих, данную модель можно использовать для отладки штатного программного обеспечения. Этот подход к моделированию технического комплекса особенно удобен и разработке «с нуля», поскольку позволяет исследовать и изменять алгоритм работы комплекса в условиях, близких к реальным.

Проанализируем более детально принципы построения стенда полунатурного моделирования для технического комплекса и требования к вычислительной аппаратуре и программному обеспечению технического комплекса и стенда.

6.1. Штатное программное обеспечение и вычислительная аппаратура объекта-оригинала

Выбор вычислительной аппаратуры, применяемой в составе технического комплекса, играет решающую роль в успехе проекта на всех этапах разработки и внедрения. Особенное внимание должно быть уделено выбору аппаратного и программного обеспечения для объектов критической инфраструктуры.

Главными критериями при выборе являются:

- отечественное производство аппаратуры и программного обеспечения;
- полнота и качество программного обеспечения;
- соответствие аппаратуры условиям штатного применения;
- широкая номенклатура интерфейсов;
- информационная и функциональная безопасность аппаратуры и программного обеспечения.

Всем перечисленным выше требованиям удовлетворяет программно-аппаратная платформа «Багет». Семейство ЭВМ «Багет» включает различные конфигурации аппаратуры, настраиваемые согласно требованиям заказчика. Программное обеспечение позволяет вести разработки с использованием метода кросс-компиляции, что повышает кибербезопасность вычислительных средств, в том числе, в составе объектов критической инфраструктуры.

При создании программно-аппаратной платформы «Багет» были использованы различные стандарты, касающиеся программного обеспечения, что повышает интероперабильность разработок, проводимых на основе данной платформы.

Программно-аппаратная платформа «Багет» является полностью отечественной разработкой.

6.2. Инструментальная среда моделирования

Инструментальная среда моделирования предназначена для разработки, исследования и эксплуатации моделей технических комплексов.

Инструментальная среда моделирования должна быть построена на следующих принципах:

- полнота функциональных возможностей для решения задачи полунатурного моделирования;
- соответствие современным методам создания, развития и сопровождения аппаратно-программных комплексов;
- обеспечение кибербезопасности.

Полнота функциональных возможностей требуется для того, чтобы создать комфортные условия для разработки модели и обеспечить сопровождение объекта-оригинала с помощью моделирующего комплекса на протяжении всего жизненного цикла, включая этапы разработки и исследования (принцип «цифрового двойника» [12]).

Функциональные возможности инструментальной среды включают: удобный пользовательский интерфейс, полноценное протоколирование, управление экспериментом, обработку данных, полученных в результате эксперимента, средства валидации и верификации моделей.

Необходимо реализовать как минимум два уровня пользовательского интерфейса – интерфейс инженера-проектировщика и разработчика модели и интерфейс инженера-исследователя, который использует комплекс для проведения экспериментов.

Следует уделить большое внимание средствам конфигурирования и настройки и обеспечить возможность выбора аппаратных и программных компонентов, включенных в эксперимент, установку начальных значений, установку единого времени на ЭВМ комплекса, настройку циклограммы работы, спецификацию параметров, накапливаемых во время работы комплекса.

Управление экспериментом включает запуск и остановку работы комплекса, средства визуального наблюдения за его функционированием, контроль процесса моделирования с точки зрения безопасности, возможность моделирования нештатных ситуаций и сбоев, которые могут возникнуть на объекте-оригинале.

Необходимы также средства обработки данных, полученных в результате эксперимента, включая визуализацию и статистический анализ.

Современные методы создания и развития программного обеспечения нацелены на обеспечение продолжительного существования инструментальной среды в быстро меняющихся

условиях [13]. Для этого необходимо использование стандартов, в том числе, стандарта HLA, средств модификации и масштабирования (расширение доступного набора интерфейсов, адаптация к новой аппаратуре и т.п.).

Необходимо также обеспечить совместимость по форматам данных с существующими внешними системами, реализующими дополнительный функционал, например, графическое представление информации, обработку big data, математические методы.

Описанный подход к разработке инструментальных средств моделирования позволит реализовать на широкий спектр задач моделирования в различных областях применения, а также задач создания цифровых двойников и обеспечение работы с интернетом вещей (включая специфические интерфейсы RFID, NFC и облачные технологии).

Кибербезопасность инструментальной среды и моделирующего комплекса в целом основана на использовании доверенных программно-аппаратных средств разработки и стандартных приемов обеспечения информационной и функциональной безопасности (авторизация, установка и контроль прав доступа, разграничение доступа, протоколирование выполняемых действий и событий и т. п.).

6.3. Математические модели компонентов объекта-оригинала

Математические модели элементов объекта-оригинала используются в случае, когда элемент физически не представлен на стенде.

Математические модели исполняются на дополнительных ЭВМ, которые не входят в состав штатной вычислительной аппаратуры объекта-оригинала (моделирующих ЭВМ, см. рис. 4). Это позволяет минимизировать влияние процесса моделирования на штатную циклограмму и создать на стенде условия функционирования штатных ЭВМ, максимально приближенные к реальным режимам эксплуатации.

Математические модели могут быть созданы с использованием различных способов формализации, например, в виде решения системы уравнений или в виде числового ряда, полученного в результате дополнительных исследований, статистических наблюдений или иными способами.

В целях расширения области применения моделирующего комплекса необходимо создавать библиотеки моделей, предназначенных для решения различных задач, а также обеспечить условия для использования ранее разработанных моделей за счет внедрения стандарта HLA.

Математические модели, имитирующие работу устройств ввода-вывода (датчиков, управляющих механизмов и прочей аппаратуры), при взаимодействии со штатной аппаратурой должны использовать протоколы передачи данных, предусмотренные для объекта-оригинала.

Инструментальная среда моделирования должна обеспечить конфигурирование совокупности математических моделей, используемых для отдельного эксперимента, настройку параметров моделирования и возможность масштабирования модели, например, увеличение числа каналов передачи данных. Необходимо также предусмотреть способы расширения стенда за счет включения дополнительных моделирующих ЭВМ.

Для того, чтобы обеспечить функционирование стенда в целом, требуется программа, управляющая процессом моделирования (программа-диспетчер). Диспетчер входит в состав инструментальной среды моделирования и инициирует запуск программ, реализующих математические модели, в соответствии с заданной циклограммой.

Для накопления данных о функционировании математических моделей следует использовать базу данных. В зависимости от временных характеристик объекта-оригинала и технических возможностей база данных может функционировать на отдельной ЭВМ, на моделирующей ЭВМ или штатной ЭВМ объекта-оригинала. Необходимо обеспечить возможность конфигурирования стенда для различных вариантов использования базы данных.

Для обеспечения визуального контроля процесса моделирования необходимо предусмотреть возможность настройки изображения, выводимого на экран монитора моделирующей ЭВМ (перечень контролируемых параметров, частота вывода, связь с программой-диспетчером и управление экспериментом).

6.4. Имитирующая аппаратура и программное обеспечение

Имитирующая аппаратура в составе стенда представлена либо натурными фрагментами объекта-оригинала (например, пульт управления оператора), либо устройствами, имитирующими внешние условия (помехи, поломки датчиков, обрывы кабеля и т.п.). Необходимо также предусмотреть возможность использования/имитации устройств промышленного интернета вещей.

Наличие имитирующей аппаратуры и соответствующего программного обеспечения в составе стенда зависит от технических возможностей и сути задач, решаемых с помощью стенда. Например, наличие штатного пульта оператора обязательно для стенда-тренажера.

Имитация внешних воздействий входит в состав средств управления экспериментом и позволяет тестировать работоспособность АСУ объекта-оригинала в случае нештатных ситуаций.

Для управления процессом моделирования отказов/сбоев/помех необходимо предусмотреть аппаратный и программный интерфейс, допускающий как ручное, так и автоматическое вмешательство в алгоритм работы стенда.

Запуск программ, имитирующих нарушение передачи данных между моделирующей ЭВМ и штатной ЭВМ, может быть либо включен в циклограмму функционирования моделирующей ЭВМ, либо выполнен вручную оператором стенда. Аналогично выполняется включение/отключение имитации помех при передаче данных.

Конфигурирование средств управления экспериментом должно выполняться через экран монитора моделирующей ЭВМ.

6.5. Аппаратура стенда полунатурного моделирования

Принципы удобства использования и гибкости конфигурации, описанные выше для программного обеспечения, также касаются аппаратуры стенда.

Необходимо предусмотреть удобные с точки зрения конструкции способы подключения, широкий набор интерфейсов, способы и средства изменения конфигурации.

7. Заключение

Несмотря на богатую историю и широкую область применения методов математического и имитационного моделирования, следует признать, что в современных условиях полунатурное моделирование, особенно для многоуровневых многокомпонентных динамических систем, завоевывает популярность и авторитет за счет использования штатного вычислительного оборудования и программного обеспечения.

Это направление моделирования активно развивается, поскольку возможности современной аппаратуры и программного обеспечения при использовании в составе стендов обеспечивают значительную экономию средств на этапах проектирования, прототипирования и испытаний.

Применение стендов для проведения экспериментов, обучения персонала и отработки нештатных ситуаций повышает надежность объекта-оригинала. Стенды полунатурного моделирования могут также служить основой для создания цифровых двойников, что особенно важно при работе с объектами критической инфраструктуры.

Методы математического моделирования не утратили своей роли, но кажутся более подходящими для детального моделирования процессов, не связанных напрямую с вычислительными средствами, например, для моделирования отдельных химических реакций, природных явлений, процессов в биологии и т.п.

Недостатком большинства реализуемых в настоящее время стендов полунатурного моделирования и систем разработки, применяемых для этого, является их уникальность и невозможность использования в других проектах. Поэтому основными тенденциями в области полунатурного моделирования в настоящее время должны стать автоматизация процесса моделирования, расширение сферы применения методов моделирования и обеспечение совместимости и повторного использования программного обеспечения за счет применения стандартов.

Продуманная параметризация системы разработки в сочетании с удобными средствами конфигурирования и настройки позволит создать программно-аппаратный «каркас», пригодный для решения широкого круга задач.

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

Tools for Developing Models of Technical Complexes

Svetlana Bazaeva, Yaroslav Zotov

Abstract. The article discusses various ways of developing models of technical complexes and analyzes the advantages and disadvantages of the applied modeling methods. On the basis of the study, the authors propose an approach to creating a modeling tool environment that allows design HIL-simulation stands for a wide range of technological processes in various application areas, operating in real time. The tools used in modeling include software and hardware.

Keywords: technical complex, HIL-simulation stand

Литература

1. ГОСТ 3.1109-82 «ЕСТД. Термины и определения основных понятий»

2. Мальков М.В., Олейник А.Г., Федоров А.М. Моделирование технологических процессов: методы и опыт // Труды Кольского научного центра РАН, 2010. № 3. С.93–101

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

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

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

6. Шеннон, Р. Имитационное моделирование систем – искусство и наука / Р. Шеннон. – М.: Мир, 1978. – 418 с.

7. Балашов В.В., Бахмуров А.Г., Волканов Д.Ю., Смелянский Р.Л., Чистолинов М.В., Ющенко Н.В. Стенд полунатурного моделирования для разработки встроенных вычислительных систем реального времени // Сборник докладов Четвертой всероссийской научно-практической конференции «Имитационное моделирование. Теория и практика» (ИММОД-2009). – Санкт-Петербург, 2009. – С. 215–219.

8. Девятков, В.В. Методология и технология имитационных исследований сложных систем: современное состояние и перспективы развития: монография / В.В. Девятков – М.: Вузовский учебник: ИНФРА-М, 2013. – 448 с.

9. КПМ Ритм [Электронный ресурс] // Режим доступа: kpm-ritm.ru (Дата обращения: 25.11.2022)

10. 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: 25.11.2022)

11.Беляев С.А., Матросов В.В. Опыт создания среды имитационного моделирования // I-methods. 2018. Т. 10. № 3. С. 14–22.

12. Гапанович Д.А., Тарасова В.А., Сухомлин В.А., Куприяновский В.П. Анализ подходов архитектурного проектирования цифровых двойников // International Journal of Open Information Technologies, 2022. Vol. 10. № 4. С. 71–81.

13. Девятков В.В. Современные системы автоматизации имитационных исследований – разработка, развитие и применение. // Сборник докладов Седьмой всероссийской научно-практической конференции «Имитационное моделирование. Теория и практика» (ИММОД-2015). Труды конференции. Ин-т проблем упр. Им. В.А. Трапезникова РАН; под общ. Ред. С.Н. Васильева, Р.М. Юсупова. 2015. С. 25–33.В.Б. Бетелин. Системы автоматизации труда программиста. М., Наука, 1990.

Пример импортозамещения программно-аппаратной архитектуры систем управления на вычислительных средствах ограниченной производительности

Т.К.Грингауз[1], А.Н.Онин[2]

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

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

Аннотация. Микроконтроллер КОМДИВ-МКЭ отечественной разработки входит в состав интерфейсных модулей семейства ИМ-АЗС. Модули предназначены для применения в качестве контроллеров технологических процессов и оборудования автозаправочной станции. На модули портирована проприетарная программа управления. Описан принцип функционирования программы. Приведен перечень ее доработок при портировании.

Ключевые слова: импортозамещение, интерфейсный модуль, программируемый логический контроллер, древовидная сеть, технологическое оборудование, АЗС, уровнемер, ТРК.

1. Введение

В рамках работ по импортозамещению программно-аппаратных комплексов разработано семейство интерфейсных модулей ИМ-АЗС. Модули предназначены для использования в составе программно-аппаратного комплекса автозаправочной станции (далее – «АЗС»). Модули применяются в качестве программируемых логических контроллеров (далее – «ПЛК») технологических процессов и оборудования АЗС. ПЛК обеспечивают управление топливно-раздаточными колонками (ТРК), подсистемой налива и контроля уровня топлива (далее – «уровнемер»), информационной стелой, передачей данных между ПЛК и управляющей информационно-кассовой подсистемой через сервер АЗС.

В состав интерфейсных модулей входит разработанный в ФГУ ФНЦ НИИСИ РАН микроконтроллер КОМДИВ-МКЭ. Первоначальное назначение микроконтроллера – мониторинг параметров линий электропитания. Производительность микроконтроллера ограниченна в связи с малым объемом оперативной памяти (128 КБ).

Программное обеспечение модуля ИМ-АЗС (далее – «программа ИМ-АЗС») разрабатывается на основе прототипа – проприетарной программы пользователя. Пользовательская программа предоставлена в виде исходного кода на языке Си. Анализ кода и описание алгоритма программы-прототипа стали неизбежным этапом разработки программы ИМ-АЗС.

В статье описан принцип функционирования программы-прототипа, выявленный реверс-инжинирингом, а также особенности его реализации в программе ИМ-АЗС.

2. Модель вычислительной системы АЗС

Модель вычислительной системы АЗС представлена на рис.1.

Вычислительная система АЗС представляет собой двухуровневую древовидную структуру, неоднородную по типу физических соединений узлов. Корнем дерева является сервер АЗС, а узлами – программируемые логические контроллеры (далее – «ПЛК»). К ПЛК подключается технологическое оборудование АЗС.

Сервер АЗС – это миникомпьютер Intel NUC (далее – «ЭВМ NUC»).

В качестве ПЛК используются интерфейсные модули семейства ИМ-АЗС (далее – «интерфейсные модули» или «модули ИМ-АЗС»).

Примечание. Термины «ПЛК» и «интерфейсный модуль» употребляются применительно к узлу в зависимости от контекста. Термин «ПЛК» обозначает роль узла в технологическом процессе. Термин «интерфейсный модуль» определяет тип устройства, реализующего узел.

На физическом уровне модули ИМ-АЗС подключаются к ЭВМ NUC по интерфейсу LIN-«токовая петля». Сервер АЗС обменивается данными с ПЛК по протоколу IDC. Канальный, транспортный и прикладной уровни модели передачи данных между узлами сети регламентируются специальным протоколом (далее – «про-

токол IDC»). Описание протокола IDC приведено в разделе 7 настоящей статьи.

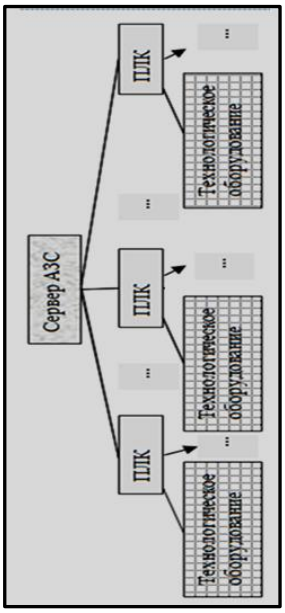

Рис. 1. Модель вычислительной системы АЗС

Под технологическим оборудованием АЗС понимаются устройства, подключаемые к ПЛК. Технологическому оборудованию соответствует третий (нижний) уровень иерархии (см. рисунок 1). ПЛК управляет подключенными к нему устройствами.

Технологическое оборудование подключается к интерфейсным модулям по последовательному интерфейсу UART (на физическом уровне поддерживаются протоколы RS-232, RS-485, DIN токовая петля) или по интерфейсу «дискретные сигналы». В статье будем рассматривать только подключение по портам UART.

Устройства классифицируются по типам. Примеры типов устройств: топливно-раздаточная колонка (ТРК), уровнемер, стела. Управление устройствами конкретного типа осуществляется посредством специфичной для этого типа системы команд. Сервер АЗС передает команды, адресованные устройству, в ПЛК. В ПЛК команды передаются по протоколу IDC (далее – «IDC-команды»).

Тип устройств подразделяется на виды. Примеры видов ТРК: WayneDresser, Топаз. Пример вида уровнемера: Струна. Протокол взаимодействия устройства с ПЛК специфичен для каждого вида устройств.

В таблице на рисунке 2 представлен перечень типов технологического оборудования, а также протоколов взаимодействия с ПЛК, поддерживаемых текущей версией программы ИМ-АЗС.

3. Состав, назначение, идентификация, конфигурация технологического оборудования АЗС

На рисунке 3 изображен пример состава и соединения основных блоков АЗС, обеспечивающих хранение топлива и его отпуск потребителям (стела на рисунке не изображена).

Подачу топлива из подземных резервуаров в бак автомобиля обеспечивают топливно-раздаточные колонки. На АЗС может быть установлено несколько топливно-раздаточных колонок.

Тип интерфейсного модуля	Тип оборудования	Интерфейс связи
ИМ-АЗС-UARTA	Уровнемер	RS-232
ИМ-АЗС-UART	Стела	RS-485
	ТРК	RS-485
ИМ-АЗС-ТП	ТРК	IEC 62056-21/ DIN токовая петля
ИМ-АЗС-ДС	ТРК	дискретные сигналы

Рис. 2.Типы технологического оборудования

Топливно-раздаточные колонки бывают односторонними и двухсторонними. Далее под ТРК подразумевается одна сторона колонки. На АЗС поддерживается сквозная нумерация ТРК.

Налив топлива происходит через рукав ТРК («пистолет»). На ТРК может быть несколько рукавов. Разные рукава одной ТРК подают разные виды топлива. Разные рукава одной ТРК не могут осуществлять налив одновременно. На АЗС поддерживается сквозная нумерация рукавов.

Топливо хранится в подземных резервуарах. В одном резервуаре хранится один вид топлива. Разные резервуары могут хранить разные виды топлива. Каждому виду топлива соответствует своя цена. На АЗС резервуары и виды топлива нумеруются сквозным образом. Таблица видов топлива с ценами отображается на стеле.

Из резервуара топливо подается в рукава ТРК. В один рукав подается топливо из одного резервуара. Из одного резервуара топливо может подаваться в несколько рукавов.

Резервуар подключен к контроллеру уровнемера (далее – «уровнемер»). К уровнемеру может быть подключено несколько резервуаров (максимальное количество зависит от вида уровнемера). В резервуаре установлено несколько датчиков. Датчик измеряет следующие параметры: объем (уровень), плотность,

температуру, давление, уровень подтоварной воды. Для вычисления объема по уровню используется калибровочная таблица. Количество датчиков, состав измеряемых параметров и калибровочная таблица конфигурируются для каждого уровнемера.

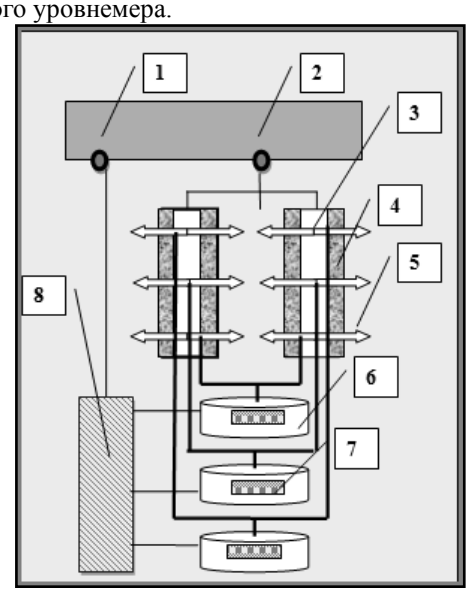

Рис. 3.Логическая схема подключения технологического оборудования АЗС.
Обозначения: 1 – интерфейсный модуль, порт UART (RS-232), 2 – интерфейсный модуль, порт UART (RS-485 или DIN-токовая петля), 3 – топливно-раздаточная колонка (двусторонняя),4 – сторона топливно-раздаточной колонки (ТРК) ,5 – рукав (пистолет) для налива топлива, 6 – резервуар с топливом, 7 – датчики, 8 – контроллер уровнемера.

Контроллер уровнемера подключается к интерфейсному модулю через порт UART (RS-232). К порту можно подключить один уровнемер.

ТРК подключаются к интерфейсному модулю через порт UART (RS-485 или DIN-«токовая» петля) через блок сопряжения. К порту может быть подключено несколько ТРК (максимальное количество зависит от вида ТРК).

Стела подключается к интерфейсному модулю через порт UART (RS-485). К порту можно подключить одну стелу.

В интерфейсных модулях ИМ-АЗС реализовано по 2 порта для подключения технологического оборудования: порты RS-485 для модулей ИМ-АЗС-UART, порты RS-232 для модулей ИМ-АЗС-UARTA, порты DIN-«токовая петля» для модулей ИМ-АЗС-ТП. Во всех исполнениях модуля ИМ-АЗС присутствует порт UART (интерфейс LIN-«токовая петля») для взаимодействия с сервером АЗС по протоколу IDC.

Программа интерфейсного модуля написана в расчете на произвольное количество портов UART. Реальное количество портов, состав технологического оборудования, взаимосвязь ТРК, рукавов, резервуаров и видов топлива описываются конфигурационными данными программы. Программа конфигурируется индивидуально для каждого интерфейсного модуля в составе вычислительной системы АЗС.

В программе поддерживаются следующие способы идентификации конфигурируемых объектов.

Для порта UART: порядковый номер при сквозной нумерации портов на интерфейсном модуле.

Для ТРК:

1) порядковый номер при сквозной нумерации на АЗС («сетевой адрес»),

2) адрес в блоке сопряжения, подключенном к порту UART («канал»),

3) порядковый номер при сквозной нумерации на модуле.

Для рукава ТРК:

1) порядковый номер при сквозной нумерации («сетевой адрес»),

2) порядковый номер рукава на стороне топливно-раздаточной колонки.

Для резервуара:

1) порядковый номер при сквозной нумерации («сетевой адрес»),

2) номер резервуара в контроллере уровнемера, подключенном к порту UART («канал»),

3) порядковый номер при сквозной нумерации на модуле.

Для видов топлива: номер вида топлива.

Для обозначения типа оборудования, подключенного к порту UART (сервер АЗС, ТРК, уровнемер, стела), используются целочисленные идентификаторы.

4. Представление конфигурационных данных в программе

Конфигурирование программы интерфейсного модуля производится при первом включении с помощью IDC-команд с сервера АЗС. Далее конфигурационные данные сохраняются в энергонезависимой памяти модуля. При последующих включениях используются сохраненные данные. Конфигурационные данные могут корректироваться в процессе эксплуатации интерфейсного модуля.

Для представления конфигурационных данных в программе используются следующие структуры.

Порты UART:

```
typedef struct _UartConfiguration
{
 uint32_t baud_rate;
```

```
    uint8_t  char_size;
    uint8_t  parity;
    uint8_t  invertTx;
    uint8_t  invertRx;
    uint8_t  timeout;
    uint8_t application; // «приложение»
                         // (тип оборудования)
  } UartConfiguration;
  staticUartConfiguration uart_configuration\
  [MAX4OF_UARTS];
```

Приложения:

```
 typedef enum _UartApplication {  …

 UART_APPLICATION_IDC_BRANCH,

//связь с сервером, протокол IDC

 …

 UART_APPLICATION_LMS, // уровнемер

 UART_APPLICATION_TRK, // ТРК

…

 UART_APPLICATION_PRICEBOARD, //
стела

 …} UartApplication;
```

```
  ТРК:
  typedef struct        _TrkConfiguration
  {
  uint8_t  trk;                  // сетевой адрес
  uint8_t  module;               // номер модуля
  uint8_t  unit;                 // номер на модуле
  uint8_t  port;                 // номер порта
  uint8_t  channel;                  // канал
  uint8_t  number_of_grades;     // число рукавов
  }
  staticTrkConfigurationtrk _configuration\
  [MAX4OF_TRK];

  Рукава ТРК:
  typedef struct _NozzleConfiguration
  {
  uint8_t   nozzle;      // сетевой адрес рукава
  uint8_t   trk;             // сетевой адрес ТРК
  uint8_t   tank;     // сетевой адрес резервуара
  uint8_t   grade;    // номер рукава на стороне
  }  NozzleConfiguration;

  static                        NozzleConfiguration\
                                nozle_configuration\
 [MAX4OF_NOZZLES];

  Резервуары:
  typedef  struct    _TankConfiguration
  {
  uint8_t        tank;        //сетевойадрес
  uint8_t    gas;         // номер вида топлива
  uint8_t    channel;       // канал
  uint8_t    port;          //номер порта
  }     TankConfiguration;

  static TankConfigurationtank_configuration\
  [MAX4OF_TANKS];

  Калибровочая таблица:
  typedef struct _TankCalibration
  {
  uint16_t      level; //уровень
  uint32_t      volume; //объем
  } TankCalibration;
  static TankCalibration calibrations\
  [MAX4OF_TANKS]\
  [CALIBRATION_TABLE_SIZE] ;

  Виды топлива:
  typedef   struct     _GasConfiguration
  {
  uint8_t   gas;     // номер вида топлива
  uint8_t   pb_row;  //  строка  таблицы стелы
  }      GasConfiguration;
  staticGasConfigurationgas_configuration\
[MAX4OF_GAS];

  Стела:
  Static PriceboardRow \
  Pricepriceboard_price_table\
  [MAX4OF_GAS];
```

5. Архитектура программы. Основные понятия и определения

5.1. Однопоточность и псевдопараллелизм

Далее будем абстрагироваться от особенностей аппаратно-программной платформы, для которой предназначена программа интерфейсного модуля.

Программа реализована как однопоточное приложение с обработкой прерываний. Состав источников прерываний зависит от типа аппаратной платформы. Прерывания могут возбуждаться аппаратными таймерами, интерфейсами UART, датчиком уровня питания.

Основной поток программы исполняет несколько «процессов» псевдопараллельно. Под процессом понимается код, предназначенный для управления конкретным типом оборудования. Процесс можно рассматривать как систему

управления однотипными единицами оборудования, подключенными к интерфейсному модулю.

В программе реализованы следующие процессы:

- branch_idc_control_process() – связь с сервером АЗС по протоколу IDC,
- trk_control_process() – управление ТРК,
- lms_control_process() – управление уровнемером,
- priceboard_control_process() – управление стелой,
- led_control_process() – управление световыми индикаторами на интерфейсном модуле.

Примечание. Управление световыми индикаторами осуществляется по интерфейсу дискретных сигналов. Процесс реализован в технологических целях, далее в статье рассматриваться не будет.

Псевдопараллельное выполнение процессов обеспечивает подпрограмма-диспетчер (dispatcher()) (см. рисунок 4).

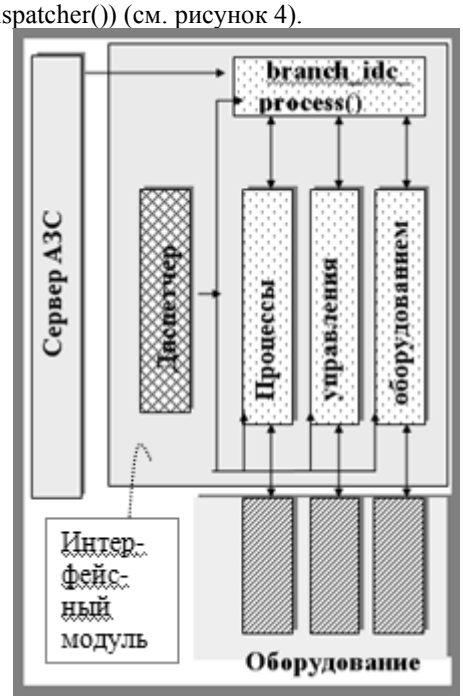

Рис. 4.Архитектура программы интерфейсного модуля

5.2. События

Под «событием» будем понимать факт, свершившийся во время выполнения программы, требующий программной реакции («обработки»). Примеры событий: 1) сигнал RESET, 2) срабатывание таймера, 3) поступление данных в порт UART, 4) падение напряжения ниже допустимого уровня.

В программе виды событий обозначаются целочисленными идентификаторами:

```
Typedef enum _OilixEvent
{
  OILIX_EVENT_RESET,…,
  OILIX_EVENT_TIMER, …,
 OILIX_EVENT_UART_RX, …,
} OilixEvent;
```

По наступлении события программа генерирует сообщение и отправляет его в очередь сообщений (далее – «очередь событий»). Тип сообщения:

```
typedef struct _OilixEventInfo
{OilixEvent    event;
  void*          user_data;
  unsigned int  id;
} OilixEventInfo;
```

(поле user_data специфицирует конкретный экземпляр события).

Поля структуры:

```
typedef struct _EventFlags {
  uint16_t   timeout;
  uint8_t    reset: 1;
  uint8_t    timer: 1;
  uint8_t    low_voltage: 1;
                                  }  EventFlags;
typedef union _EventInfo {
          EventFlags   flags;
          uint32_t     word;
                                 } EventInfo;
```

Создают сообщения и отправляют их в очередь событий основной поток программы и обработчики прерываний. Вычитывает события из очереди только основной поток (диспетчер).

5.3. Процессы

Процессы реализованы в виде функций – обработчиков событий:

```
typedef void (*ProcessHandler)\
(OilixEventInfo* event_info);
```

Каждый процесс обрабатывает свой набор событий. Процесс описывается следующим типом данных:

```
typedef struct _ProcessInfo   {
  ProcessHandler      handler;
  EventInfo             events;}  ProcessInfo;
```

Основной поток программы при запуске формирует список процессов:

```
ProcessInfo *process_list[] =
{
   &led_control_process_info,
   &lms_control_process_info,
    &trk_topaz_control_process_info,
    …..
     &branch_idc_process_info,
};
```

Далее список процессов обрабатывается диспетчером.

5.4. Диспетчер

Диспетчер выполняет бесконечный цикл. В теле цикла:

- вычитывает сообщение из очереди событий,
- последовательно вызывает все обработчики события в соответствии со списком процессов, передавая им вычитанное событие в качестве параметра.

Обработчики событий игнорируют события, не предназначенные своему процессу. Таким образом, можно считать, что каждый процесс работает циклически, асинхронно по отношению к другим процессам, в каждой итерации обрабатывая очередное предназначенное ему событие. Программу интерфейсного модуля можно представлять себе как множество параллельных событийно-управляемых систем [1].

На каждой итерации диспетчер анализирует уровень напряжения. При падении напряжения ниже критического уровня диспетчер сохраняет состояние процессов в энергонезависимой памяти (см. раздел 8 настоящей статьи).

5.5. Обмен данными между процессами

Процессы хранят свои конфигурационные данные и данные о состоянии управляемых объектов в статической области памяти. Доступ к этим данным из других процессов осуществляется посредством глобальных интерфейсных функций. Библиотеки интерфейсных функций разработаны для всех типов оборудования, поддерживаемых программой.

5.6. Диверсификация протоколов управления

Для каждого типа технологического оборудования существуют разные аппаратные реализации. Примеры видов ТРК: Wayne Dresser, Топаз. Протокол взаимодействия устройства с интерфейсным модулем специфичен для каждого вида устройств. Предполагается, что на одном интерфейсном модуле для всех однотипных единиц оборудования используется один и тот же протокол управления. На разных экземплярах интерфейсного модуля протоколы для одного и того же типа оборудования могут различаться. Протокольно-зависимые фрагменты кода выделены в отдельные программные модули. Выбор протокола осуществляется условной компиляцией.

6. Управление оборудованием с интерфейсного модуля по портам UART

6.1. Обобщенная постановка задачи управления для разных типов оборудования

Проанализированы программные тексты процессов для разных типов оборудования (уровнемер, ТРК, стела). Каждый процесс реализован в отдельном наборе программных модулей, тексты для разных процессов не зависят друг от друга. Тем не менее, можно выделить в них общие черты, сходные по способу реализации. Сходство реализаций предопределено сходством условий при постановке задач управления. Приведем типовые условия задачи.

Будем рассматривать процесс как систему управления (СУ) с несколькими (MAX_ITEMS) объектами управления. Объект управления – единица оборудования (все единицы однотипны). СУ может управлять также отдельными элементами объектов управления. Примеры: СУ ТРК управляет сторонами топливно-раздаточных колонок и рукавами налива топлива, СУ уровнемера управляет уровнемером и подключенными к нему резервуарами.

Объекты управления подключены к портам UART. К одному порту через общий контроллер по шине может быть подключено несколько объектов. У каждого объекта управления (или его управляемого элемента) есть свой уникальный адрес в контроллере.

На модуле несколько (MAX_PORTS) портов UART. Только часть из них (NUMBER_OF_EXCHANGE_CHANNELS) используется для подключения объектов управления.

Протокол управления определяет состав, последовательность и формат запросов (команд) от СУ к объекту, а также состав и формат допустимых ответов на каждый запрос.

Формат сообщений, передаваемых по последовательному каналу, специфицирует маркеры начала и конца сообщения. В составе сообщения передается контрольная сумма. Ответ объекта состоит из короткого сообщения, за которым могут передаваться данные. Короткое сообщение подтверждает или опровергает успешность приема команды СУ. Данные ответа передаются только в случае успешного приема запроса.

Протокол определяет максимальное время задержки ответа оборудования, а также предельно допустимое число повторных посылок запроса.

Задачи СУ различаются для разных типов оборудования.

СУ уровнемера должна выполнять мониторинг состояния резервуаров с сохранением текущего состояния.

СУ ТРК должна управлять наливом топлива по рукавам ТРК, подключенных к модулю.

СУ стелы должна обеспечивать отображение актуальных значений цены видов топлива.

Мы будем рассматривать обобщенную постановку задачи, в соответствии с которой СУ должна:

- отправлять последовательность запросов объектам управления по сценарию, определенному протоколом,

- принимать и интерпретировать ответы оборудования,

- извлекать из ответов и сохранять данные о текущем состоянии объектов управления.

Данные для запросов и текущее состояние объектов управления должны храниться в статической памяти процесса. Другим процессам должна быть обеспечена возможность изменения и опроса перечисленных данных.

6.2. События процесса

В текущей версии программы процессы обрабатывают следующие события:

OILIX_EVENT_RESET,
OILIX_EVENT_TIMER,
OILIX_EVENT_UART_RX.

Событие OILIX_EVENT_RESET обрабатывается всеми процессами. По этому событию процесс конфигурирует и инициализирует свои структуры.

Событие OILIX_EVENT_TIMER обрабатывается только в том случае, если оно порождено таймером процесса. С каждым процессом связан таймер, отмеряющий тайм-аут CONTROL_TIMEOUT. С процессом ТРК связаны два таймера, отмеряющие тайм-ауты: STOP_TIMEOUT, CONTROL_TIMEOUT. Таймеры всех процессов пронумерованы сквозным образом. Для маркировки каждого таймера отведен свой бит в поле user_data структуры события. Индексы битов и величины таймаутов для каждого процесса устанавливаются при инициализации. Таймер «STOP_TIMEOUT» тактирует сценарий перевода объекта управления в состояние OFF_LINE. Таймер «CONTROL_TIMEOUT» тактирует рабочий цикл процесса.

Событие OILIX_EVENT_UART_RX обрабатывается процессом branch_idc_process() и процессом уровнемера. Событие возникает при появлении новых данных во входном буфере порта UART. Номер порта передается в поле user_data структуры события.

6.3. Структуры данных процесса

Процесс использует следующие типы структур данных:

- конфигурационные данные (см. раздел 4 настоящей статьи);

- программное представление объектов управления (единиц оборудования) в их текущем состоянии (UNITS, далее - «юниты»);

- каналы обмена данными по портам UART (EXCHANGE_CHANNELS, далее – «каналы управления»).

Перечисленные данные хранятся в виде массивов структур в статической памяти процесса. Доступ к конфигурационным данным и к юнитам предоставлен другим процессам через интерфейсные функции.

Тип структуры юнита описывает тип оборудования и определяется индивидуально для каждого типа. Тип структуры канала управления зависит от вида оборудования (т. е. от протокола управления) и определяется индивидуально для каждого протокола.

Примеры:

1) Юниты процесса уровнемера:

```
typedef struct _LMSInfo {
      uint8_t tank;
                              …} LMSInfo;
static LMSInfo \
lms_unit[MAX4OF_TANKS];
```

2) Юниты процесса ТРК:

```
typedef  struct _TRKInfo{
uint8_t   trk;
                           …} TRKInfo;
static TRKInfo \
lms_unit[MAX4OF_TRK];
```

3)Канал управления для уровнемера «Струна»:

```
typedef struct _StrunaExchangeInfo
                                 {…….
                                  } StrunaExchangeInfo;
static StrunaExchangeInfo strunaExchangeInfo
[LMS4OF_EXCHANGE_\
CHANNELS];
```

Упрощенное изображение связей между юнитами, каналами управления, портами UART и управляемыми объектами приведено на рисунке 5.

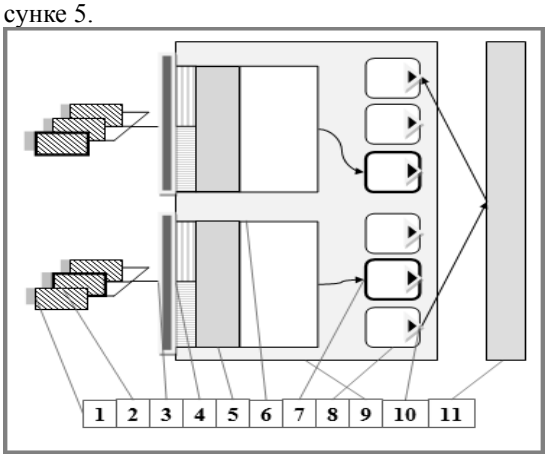

Рис. 5. Управление технологическим оборудованием. Структуры данных процесса. Обозначения: 1 – единицы оборудования, 2 –активный объект управления, 3 – порт UART, 4 – буферы UART (rx, tx), 5 – буфер данных канала, 6 – канал управления, 7 – активный юнит в канале управления, 8 – юниты, 9 – процесс управления, 10 – индикаторы изменения состояния юнита, 11 – branch_idc_process()

Процесс инициализирует массивы юнитов и каналов управления на основе конфигурационных данных. В массиве юнитов регистрируются только те единицы оборудования, которые должны быть подключены к модулю. Индекс юнита в массиве – порядковый номер единицы оборудования на модуле. В массиве каналов управления регистрируются только те порты, к которым подключается оборудование, обслуживаемое процессом. Такие порты опознаются по полю application в конфигурации порта.

Содержимое юнитов и каналов управления изменяется по мере работы программы и отражает текущее состояние объектов управления. Корректирует юниты сам процесс, а также процесс branch-idc_process(). Процесс branch-idc_process() обеспечивает взаимодействие сервера АЗС с процессом управления оборудованием. Процессы изменяют и вычитывают содержимое юнитов асинхронно.

Поля юнита:

- идентифицируют объект управления;
- хранят текущие значения параметров состояние объекта;
- позволяют узнать об изменении состояния объекта.

Различия в структуре каналов управления для уровнемера и для ТРК существенны. Приведем обобщенное описание информационного наполнения канала.

Канал управления должен хранить, идентифицировать или ссылаться на следующие сущности:

- номер порта UART,
- конфигурационные данные порта UART,
- активный объект управления,
- состояние канала (состояние активного объекта),
- код текущего запроса,
- входной и выходной буферы порта UART,
- буфер хранения ответа оборудования,
- состояние приема данных из порта UART,
- счетчик ошибок.

Под активным объектом управления понимается единица оборудования, с которой взаимодействует канал управления в текущий момент. Пока не получен ответ на текущий запрос, процесс не может отправлять запросы другим объектам по тому же каналу управления. Канал управления может содержать ссылку на юнит или использовать другой способ идентификации активного объекта.

При входе в очередную итерацию рабочего цикла канал находится в одном из двух состояний: «не активен», «ожидает ответа».

Под текущим запросом понимается код последнего выполненного запроса по протоколу. В протоколе управления каждой команде присвоен код. По сценарию управления очередной запрос формируется с учетом предыдущего.

Под состоянием приема данных понимается состояние функции побайтового чтения входного буфера порта UART, используемой для приема ответа оборудования. Функция работает по принципу конечного автомата. Примеры состояний, в которых может остаться автомат по завершении текущей итерации процесса: «ожидание подтверждения», «ожидание старт-символа», «ожидание стоп-символа» (точный перечень состояний специфицирован протоколом управления).

6.4. Обмен данными по каналу управления

Выбор канала управления зависит от того, по какому событию запустилась очередная итерация рабочего цикла процесса.

Если итерация инициирована таймером (событие OILIX_EVENT_TIMER), то в ней последовательно обрабатываются все каналы, обслуживаемые процессом. Каналы опознаются по значению поля application в конфигурации порта. Если в канале нет активного объекта, то процесс отправляет в него запрос с поиском активного объекта, после чего переходит к следующему каналу.

Если итерация инициирована событием OILIX_EVENT_UART_RX, то она обрабатывает только тот канал, по которому пришли данные. Канал выбирается по номеру порта, указанному в поле user_data структуры события. Активен тот объект, от которого пришли данные.

Далее считаем, что канал и объект выбраны. Дальнейшая обработка состоит из следующих шагов.

Шаг 1. Прием ответа объекта управления. Если ответ получен, то переход к шагу 2, иначе – переход к шагу 4.

Ответ считается полученным, если в результате побайтового чтения входного потока данных в буфере ответа оборудования накопился пакет, который соответствует формату положительного ответа по протоколу управления.

Шаг 2. Интерпретация ответа.

Ответ интерпретируется в соответствии с протоколом по коду текущего запроса. Изменение состояния объекта фиксируется в юните.

Шаг 3. Формирование следующего запроса по протоколу. Переход к шагу 6.

Шаг 4. Обработка счетчика ошибок.

Если исчерпано число допустимых ошибок, или завершена последовательность запросов по протоколу, объект сделать неактивным.

Шаг 5. Отправка следующего запроса.

Отправка следующего запроса предполагает, что если текущий объект неактивен, нужно перейти к следующему объекту в том же канале.

Шаг 6. Завершение работы с каналом в текущей итерации.

7. Связь с сервером АЗС по протоколу IDC

7.1. Роль сервера АЗС в управлении технологическим оборудованием

Программа интерфейсного модуля управляет технологическим оборудованием независимо от того, подключен ли модуль к серверу АЗС. Мониторинг состояния оборудования может осуществляться в автономном режиме. Участие сервера в управлении необходимо в следующих случаях:

- для передачи команд оператора,
- для мониторинга состояния оборудования с сервера в автоматическом режиме.

Команды оператора необходимы для конфигурирования оборудования и для управления технологическими процессами в ручном режиме. Пример: санкционирование налива топлива после оплаты.

Команды передаются в интерфейсный модуль с сервера АЗС. Сервер АЗС взаимодействует с интерфейсными модулями по протоколу IDC.

7.2. Протокол IDC. Основные понятия и определения

Протокол IDC определяет правила передачи сообщений в древовидной, разнородной по типу соединений сети ПЛК (далее – «IDC-сеть»). IDC-сеть предназначена для организации управления технологическим оборудованием с сервера АЗС.

ПЛК рассматривается как узел древовидной сети (далее – «IDC-узел»). Север АЗС выполняет роль корневого узла. Каждому узлу соответствует уникальный числовой идентификатор (далее – «IDC-адрес»).

Узлы подразделяются на управляемые и управляющие. Управляющий узел передает команды управляемым узлам и принимает ответы. Команды и ответы передаются в «IDC-формате» (формат описан ниже).

Узел обладает набором «IDC-интерфейсов». Под IDC-интерфейсом понимается набор IDC-команд, предназначенных для выполнения узлом какой-либо задачи. IDC-интерфейс идентифицируется уникальным номером в рамках сети. Разные узлы могут обладать одним и тем же программным интерфейсом. Каждой команде программного интерфейса соответствует уникальный в рамках этого интерфейса номер.

IDC-формат сообщения определен для канального, транспортного и прикладного уровней модели передачи данных.

Формат кадра для канального уровня: [«Старт» «Данные» «Длина» «Контрольная сумма» «Стоп»].

Все данные между символами «Старт» и «Стоп» не должны совпадать с этими символами.

Формат пакетов транспортного уровня: [«адрес узла отправителя» «адрес узла получателя» «идентификатор интерфейса» «команда интерфейса»].

Формат команды программного интерфейса (прикладной уровень): [«номер команды»][«данные команды»].

«Данные команды» – это байтовый массив. Интерпретирует данные программный интерфейс в соответствии со специфичным для него протоколом.

7.3 Процесс branch_idc_control _process()

На интерфейсном модуле прием и отправку пакетов в формате IDC выполняет процесс branch_idc_control_process() (далее – branch_idc). Для связи по IDC используется порт UART, в конфигурации которого в поле «application» указано значение UART_APPLICATION_IDC_BRANCH.

В программе интерфейсного модуля реализованы IDC-интерфейсы к поддерживаемым типам технологического оборудовании, а также к двум программным сервисам: dm (менеджер данных), azs-config (сервис конфигурирования).

Для описания IDC-интерфейса предназначен следующий тип данных:

```
typedef struct _IDCInterface {uint8_t          id;
FuncIdcGetPendingRequest    pending_request;
FuncIdcHandleData               handle_data;
                              } IDCInterface;
```

Поля структуры:

- id – номер интерфейса;
- pending_request – функция поиска готового ответа;
- handle_data – функция-обработчик IDC-пакета.

Формат функции – handle_data:

```
typedef void (*FuncIdcHandleData)(uint8_t
source, const uint8_t* buffer, uint8_t size);
```

Формат функции pending_request:

```
typedef struct _IDCReply {
    uint8_t    packet_size;
    uint8_t packet[IDC_MAX_REPLY_LENGTH];
                              } IDCReply;
#define    IDCRequest          IDCReply

typedef IDCRequest* (*FuncIdcGetPendingRe-
quest)(void);
```

В статической памяти определен список интерфейсов:

```
static IDCInterface* idc_interface_list[] =
{
 &idc_dm_interface,
 &idc_lms_control_interface,
 &idc_trk_control_interface,
 &idc_azs_config_interface,
 &idc_priceboard_interface,
};
```

Каждый интерфейс реализован в своем программном модуле. Модули интерфейсов включаются в программный модуль процесса branch_idc:

```
#include <idc2/idc-common.h>
#include <idc2/idc-dm.c.h>
#include <idc2/idc-azs-config.c.h>
#include <idc2/idc-lms-control.c.h>
#include <idc2/idc-trk-control.c.h>
#include <idc2/idc- priceboard.control.c.h>
```

Для приема запросов и формирования ответов используется буфер в статической памяти (IDC-буфер). Для буфера определены смещения в соответствии с форматом IDC-пакета:

```
typedef enum _IDCOffset
{
IDC_OFFSET_INTERFACE,
IDC_OFFSET_SOURCE,
IDC_OFFSET_DESTINATION,
IDC_OFFSET_COMMAND,
IDC_OFFSET_DATA,
}
```

Процесс branch_idc обрабатывает события OILIX_EVENT_UART_RX, возбуждаемые по приходе очередной порции данных в UART-порт процесса. Запускается функция побайтового чтения входного буфера порта. Функция вычленяет из входного потока кадры канального уровня протокола IDC, извлекает пакеты транспортного уровня, помещая их в IDC-буфер. Функция работает по принципу конечного автомата. Если по исчерпании входных данных порта в IDC-буфере не собрался корректный пакет, или адрес узла получателя не совпал с адресом модуля, итерация процесса branch_idc завершается. В противном случае вызывается функция интерпретации пакета. Функция интерпретации вычитывает из IDC-буфера номер интерфейса и запускает соответствующий обработчик сообщений handle_data из списка интерфейсов.

Обработчик handle_data вычитывает из IDC-сообщения команду и данные и выполняет команду в соответствии с ее семантикой. Под выполнением команды понимается внесение изменений в юниты или в конфигурационные данные процессов управления оборудованием. Изменения вносятся путем вызова интерфейсных функции процессов.

До объекта управления команда дойдет асинхронно. Соответствующая ей последовательность команд будет выработана процессом управления оборудованием. Команды будут формироваться в соответствии с протоколом управления с учетом измененных данных юнита. Сопутствующие изменения состояния объекта управления будут зафиксированы в юните.

По завершении обработчика handle_data процесс branch_idc формирует и отправляет ответ источнику IDC-запроса (т. е. серверу АЗС). В ответе отправляются сведения об очередном изменении состояния технологического оборудования. Ответ формируется следующим образом.

Процесс branch_idc находит в списке интерфейсов очередной интерфейс, по которому произошли изменения. Об изменениях сообщает функция pending_request интерфейса. Функция pending_request ищет очередной измененный элемент в массиве юнитов для своего типа оборудования. В случае успеха функция формирует ответ. В ответе передаются параметры измененного состояния. Если ни один интерфейс не сообщил об изменениях, в ответе отправляется сообщение «NO_DATA».

8. Сохранение состояния в энергонезависимой памяти

Программа интерфейсного модуля восстанавливает состояние ПЛК после перезагрузки. Это достигается следующим образом.

Состояние ПЛК описывают посредством «retain-переменных». Понятие «retain-переменные» используется для обозначения переменных, которые сохраняют значение при отключении питания контроллера. Между сеансами работы контроллера retain-переменные сохраняются в энергонезависимой памяти.

В программе интерфейсного модуля retain-переменные объявляются с атрибутом (section (”mram”):

__attribute__ ((section (”mram”), aligned (32)))

Компилятор размещает переменные, объявленные с этим атрибутом, в секции «mram» и помещает адреса начала и конца секции в переменные

unsigned char __start_mram;

unsigned char __stop_mram;

С атрибутом (section (”mram”)) объявляются юниты и конфигурационные данные процессов.

Для сохранения retain-переменных предназначена энергонезависимая память EEPROM объемом 1 Мбайт.

При падении питания ниже критического

уровня программа сохраняет секцию «mram» в энергонезависимой памяти.

По включении ПЛК программа восстанавливает содержимое секции из энергонезависимой памяти.

9. Портирование программы на аппаратную платформу ИМ-АЗС

Особенности аппаратной платформы модуля ИМ_АЗС привели к необходимости внесения следующих изменений в текст программы.

В связи с отсутствием операционной системы для процессора КОМДИВ-МКЭ функции технологической обвязки, обеспечивающие доступ к базовым интерфейсам, включены в текст программы.

Разработана и включена в текст программы библиотека функций асинхронного обмена данными по интерфейсу UART в соответствии со стандартом POSIX.

В связи с малым объем ОЗУ (128 КБ) программный код выполняется из РПЗУ модуля. Для достижения требуемого быстродействия размещение секций кода оптимизировано за счет размещения обработчиков событий в ОЗУ.

В связи с отсутствием в процессоре КОМДИВ-МКЭ сопроцессора плавающей арифметики для выполнения операций с плавающей точкой используется библиотека (GNU) SoftFloat. Библиотека SoftFloat подключается при компиляции программы [2] с использованием компилятора gcc.

Диагностика падения питания осуществляется путем опроса датчика питания.

10. Заключение

Проведен анализ текста проприетарной программы управления технологическим оборудованием АЗС. Выявлены и описаны архитектура, базовые примитивы и принцип функционирования программы.

Выполнено портирование программы на модули семейства ИМ-АЗС. В статье приведен перечень сопутствующих доработок.

Замечена схожесть подходов в реализации процессов управления разными типами оборудования. Это создает предпосылки для последующей оптимизации программы на основе объектного или автоматного программирования [1].

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

Control System Software and Hardware Architecture Import Substitution on Limited-Performance Computing Facilities

T.K. Gringauz, A.N. Onin

Abstract. The microcontroller KOMDIV-MKE of domestic design is part of the interface modules of the IM-AZS family. The modules are designed for use as technological process and equipment controllers of a gas filling station. A proprietary control program has been ported to the modules. The program operation principle is described. A list of its improvements during porting is given.

Keywords: import substitution, interface module, programmable logic controller, tree network, technological equipment, filling station, level gauge, fuel dispenser

Литература

1. Н.И. Поликарпова, А.А.Шалыто. Автоматное программирование. Санкт-Петербург, Санкт-Петербургский Государственный Университет Информационных Технологий Механики и Оптики, 2008.

2. В.А.Галатенко, Г.Л.Левченкова, С.В.Самборский. Особенности сборки кросс-компилятора GCC и бинарных утилит. «Труды НИИСИ РАН», Т. 12 (2022), № 4, с. 43-49.

Обзор существующих решений в области моделирования технологических процессов

Я.А. Зотов[1]

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

Аннотация. Статья содержит обзор некоторых существующих решений в области моделирования в некоторых отраслях. Приведены определения цифровых двойников. Даны описания программ, реализующих функционал цифровых двойников, в том числе с открытым исходным кодом. Описаны некоторые математические решения.

Ключевые слова: моделирование, цифровой двойник, автоматизированные системы управления

1. Введение

Разработка и отладка программного обеспечения автоматизированных систем управления технологическими процессами должна базироваться на модели, адекватно учитывающей основные параметры и алгоритмы функционирования объекта автоматизации.

К надёжности функционирования и к функциональной безопасности программного обеспечения систем управления сложными техническими процессами предъявляются повышенные требования. В связи с этим при разработке программного обеспечения необходимо применять модели, отражающие существенные и наиболее важные характеристики функционирования объекта автоматизации.

Использование функциональных моделей объекта автоматизации необходимо в тех случаях, когда оценку корректности системы управления невозможно осуществить в условиях действующего объекта автоматизации в связи с ущербом, который может быть нанесен системой управления из-за ошибок в прикладном программном обеспечении.

Существует большое количество средств моделирования технологических процессов, применяемых в настоящее время. Такие средства основаны на математических моделях, системах искусственного интеллекта, машинного обучения и технологии цифровых двойников.

2. Цифровые двойники

2.1. Определения цифровых двойников

Формальные идеи цифровых двойников возникли в двухтысячные годы [1]. Однако, долгое время не существовало ни строго определения этого понятия, ни технической возможности реализации модели такого уровня сложности. Приведем несколько определений цифровых двойников:

Цифровой двойник — это интегрированная мультифизическая, многомасштабная вероятностная симуляция готового транспортного средства или системы, в которой используются наилучшие доступные физические модели, обновления датчиков, история парка и т. д., чтобы отразить жизнь соответствующего летающего двойника. NASA, 2012 [2].

Цифровой двойник — это компьютеризированная модель физического устройства или системы, в которой представлены все функциональные особенности и связи с рабочими элементами. Чен, 2017 [3]

Цифровой двойник представляет собой живую модель физического актива или системы, которая постоянно адаптируется к операционным изменениям на основе собранных онлайн-данных и информации и может прогнозировать будущее соответствующего физического аналога. Лю и др., 2018 [4].

Цифровой двойник — это набор виртуальной информации, которая полностью описывает потенциальное или реальное физическое производство от микроатомного до макрогеометрического уровня. Чжэн и соавторы, 2018 [5].

Цифровой двойник — это цифровое представление физического предмета или сборки с использованием интегрированных симуляций и сервисных данных. Цифровое представление содержит информацию из нескольких источников на протяжении всего жизненного цикла продукта. Эта информация постоянно обновляется и визуализируется различными способами для прогнозирования текущих и будущих условий как в проектной, так и в операционной среде, чтобы улучшить процесс принятия решений. Врабич и др., 2018 [6].

Цифровой двойник — это виртуальный экземпляр физической системы (двойника), который постоянно обновляется данными о производительности, обслуживании и состоянии последней на протяжении всего жизненного цикла физической системы. Манди, 2019 [7].

В 2021 был издан ГОСТ Р 57700.37-2021 «Компьютерные модели и моделирование. Цифровые двойники изделий. Общие положения», содержащий следующее определение. Цифровой двойник изделия — система, состоящая из цифровой модели изделия и двусторонних информационных связей с изделием (при наличии изделия) и (или) его составными частями [8].

2.2. GE Predix

Predix — это облачная платформа для промышленного интернета, разработанная компанией General Electric для анализа больших данных в промышленности с учетом соблюдения всех требований кибербезопасности [9]. Датчики, установленные на подключенном к системе оборудовании, собирают большие объемы информации, хранящиеся в единой защищенной облачной платформе. Программное обеспечение на базе Predix использует систему предиктивной аналитики для изучения данных, обнаружения и диагностики неисправностей, что повышает надежность и доступность активов, одновременно снижая затраты на эксплуатацию и обслуживание. Predix обеспечивает повышение эффективности благодаря объединению разрозненных источников данных в единое информационное пространство и использованию передовых средств анализа для принятия правильных решений.

Наиболее мощной частью Predix является анализ больших данных на основе цифровых двойников, при котором различные исходные состояния физического оборудования собираются и хранятся в виртуальном информационном пространстве, а оборудование управляется путем построения точных моделей и прогнозов поведения.

Predix содержит каталог моделей, разработанных компанией General Electric и ее партнерами в форме программных интерфейсов приложений (API), и предоставляет тестовые данные, чтобы пользователи могли быстро использовать существующие модели для обучения своих моделей. Разработанные пользователями модели также могут публиковаться в этом каталоге моделей и использоваться другими пользователями. Данные модели включают в себя обнаружение аномалий, анализ, обработку сигналов, контроль качества и оптимизацию работы. В соответствии с широко признанными типами анализа больших промышленных данных, анализ можно разделить на диагностический, прогнозирующий и превентивный.

В дополнение к этим моделям анализа, General Electric предоставляет более 300 моделей активов и процессов. Эти модели связаны с различными продуктами General Electric, включающими различные атрибуты и 3D-модели, удобные для пользователей для быстрого создания цифровых двойников.

Примеры использования системы Predix:

- прогнозный анализ работы оборудования в режиме реального времени;
- превентивная аналитика;
- оперативное управление производством;
- хранение и обработка технологических данных;
- оптимизация технологического процесса;
- техническое обслуживание и ремонт промышленного оборудования.

2.3. ThingWorx

ThingWorx — платформа, предназначенная для сборки и запуска компонентов промышленного интернета вещей. С её помощью можно сократить временные, финансовые затраты на ремонт и снизить риски выхода оборудования из строя [10].

ThingWorx предоставляет возможность быстрой сборки и запуска сетевых приложений для реализации новых возможностей в стремительно развивающемся мире интернета вещей с помощью цифровых двойников. Цифровые двойники содержат информацию, описывающую изделие и находящуюся в разных системах учета [11].

Платформа ThingWorx Converge взаимодействует с такими системами через реализованные в них интерфейсы программирования приложений API. Она понимает взаимосвязь между изделиями и информацией о них, хранящейся в других системах. Converge выступает посредником между цифровым двойником и приложениями, которые им пользуются. Приложения получают информацию от цифровых двойников и могут управлять ими через предоставленный API.

Совместно с ThingWorx может использоваться решение ColdLight, позволяющее проводить прогнозирующий анализ собранных данных и фильтровать их для дальнейшей обработки с использованием методов интеллектуального анализа. Эта система дает доступ ко всем данным, необходимым для получения представления обо всех доступных возможностях, благодаря чему можно принять информированное решение.

2.4. Автодискавери

Программно-аппаратный комплекс "Авто-

дискавери", созданный проектом "Цифровые дороги", является системой узкого назначения. Он предназначен для создания цифровых двойников и контроля состояния объектов дорожно-транспортной инфраструктуры. Это решение позволяет получать актуальные данные о состоянии объектов дорожно-транспортной инфраструктуры, принимать оперативные меры по восстановлению поврежденных объектов, а также в автоматизированном режиме готовить необходимую документацию [12].

Мобильные лаборатории комплекса проводят автоматизированную оцифровку объектов дорожно-транспортной инфраструктуры при проезде с точными измерениями координат и параметров объектов и панорамными фото. Благодаря разработкам на базе машинного зрения, нейросетевого анализа, LiDAR-технологиям система с точностью позиционирования 10 см в режиме онлайн определяет и классифицирует объекты на дороге — светофоры, дорожные знаки, дорожное полотно, и их текущее состояние. В результате получается полное документирование, картографическая подложка и паспорта объектов со всей историей и актуальными данными.

Система "Автодискавери" полностью автоматизирована, благодаря чему исключается человеческий фактор и повышается производительность оцифровки, так как актуальные параметры объектов структурируются и заносятся в базу во время проезда без долгой ручной обработки. Производительность одной мобильной лаборатории — 150 км в день. Оцифровка города с протяженностью улично-дорожной сети до 1000 км и преимущественно двухполосным движением по каждому из направлений занимает не более 2 недель.

2.5. Zenkit

Проект Zenkit интересен тем, что он представляет собой программное обеспечение, поддерживающее создание цифровых близнецов для рабочих процессов компании. Данное цифровое решение создает точную копию бизнеса и позволяет отслеживать результаты развития в реальном времени [13].

Zenkit предоставляет возможность использования только необходимых функций программы. Пользователь сам определяет, какую информацию он хочет отслеживать, как и где ее хранить, и кто может получить к ней доступ.

С помощью произвольных полей можно объединить сразу несколько элементов компании воедино: клиентские данные, документацию, счета, проекты, ресурсы, события и т.д. Технология Zenkit позволяет соединить данные аспекты и эффективно использовать их совместный потенциал. Имеются различные режимы отображения данных: календарь для планирования задач, канбан-доска для индивидуальных заданий, таблица для отслеживания важных показателей с помощью формул.

Zenkit может стать цифровым двойником для любой компании, команды или цели. В нем есть возможность управления проектами в режиме реального времени, задания участникам рабочих пространств, обсуждения работы, используя функции комментирования и упоминания. Возможно использование Zenkit для хранения данных и помощи клиентам, отслеживания потенциальных клиентов, управления воронкой продаж и обеспечения клиентам прямого доступ к их счетам, проектам и системе коммуникации.

В Zenkit можно загружать документы и изображения, хранящиеся на компьютере или в любом облачном хранилище, открыть к ним доступ для коллег для совместной работы над проектами.

Также есть возможность создания и управления базой данных сотрудников компании. В Zenkit можно настроить канал для внутренней коммуникации, подготовить ресурсы для ознакомления новых сотрудников с рабочими процессами, управлять заявлениями о приеме на работу, назначать зарплату и т.д.

2.6. Ansys Twin Builder

Ansys Twin Builder — это инструмент для создания, валидации и развертывания цифровых двойников и предиктивного обслуживания производственных активов [14].

Цифровой двойник применяется на всех стадиях жизненного цикла изделия. На этапе эскизного проектирования создаются различные варианты модели и выбирается наиболее оптимальное техническое решение. Далее на этапе технического проектирования созданная модель дорабатывается с учетом взаимодействия всех компонентов системы, режимов работы ПО и условий окружающей среды. На этапе изготовления происходит определение допусков и требований для обеспечения желаемых характеристик и безотказной работы изделия. Получаемые во время эксплуатации цифрового двойника данные обеспечивают своевременную диагностику системы и перекалибровку ее параметров с учетом новых потребностей потребителя.

Предиктивная аналитика данных, получаемых в режиме реального времени от виртуальных датчиков, обеспечивается за счет удобной интеграции Ansys Twin Builder с платформами промышленного интернета вещей.

Разработка цифрового двойника Ansys Twin

Builder ускоряется благодаря использованию готовых моделей и элементов:

- стандартные языки и форматы обмена данными: VHDL-AMS (IEEE 1076.1), Modelica, SML (Simplorer Modeling Language), FMI (Functional Mock-up Interface), C/C ++, SPICE;
- обширные библиотеки 0D-элементов для разных областей применения;
- 3D-решатели и модели пониженного порядка (ROM);
- импорт моделей, созданных в Simulink;
- экспорт созданных на Modelica моделей в формате FMU;
- использование SCADE Suite для верификации, оптимизации и калибровки параметров безопасности критически важного встроенного программного обеспечения.

Эффективность валидации и оптимизации характеристик системы повышается благодаря следующим характеристикам:

- эффективный алгоритм синхронизации и адаптивной настройки решателей на каждом временном шаге;
- подключение к высокопроизводительным вычислительным ресурсам;
- продолжение расчета после сохранения проекта;
- изменение значений переменных в режиме реального времени для улучшения характеристик модели;
- использование Ansys DesignXplorer для выполнения сложной параметрической оптимизации;
- использование SCADE Display для моделирования интерактивного человеко-машинного интерфейса (HMI);
- создание сценариев на Python или Visual Basic для автоматизации процессов моделирования;
- создание пользовательских инструментов с API-интерфейсом для Python;
- подключение новых библиотек моделей и расширений из магазина Ansys ACT.

Повышение производительности достигается за счет следующих возможностей

- быстрое подключение к поддерживаемым платформам промышленного интернета вещей (IIoT);
- экспорт моделей для создания межплатформенной, облачной инфраструктуры цифровых двойников для IIoT-платформ SAP, PTC или GE Predix;
- большой выбор графических и табличных отчетов для визуализации результатов моделирования;
- быстрый экспорт диаграмм, графиков и таблиц в Microsoft Excel или в различные форматы файлов изображений;
- автоматическое создание сводных отчетов по результатам моделирования в HTML и PDF.

2.7. Решения Siemens

Создание цифрового двойника производства с помощью средств Siemens на примере производства напитка [15].

На первом этапе происходит создание цифрового двойника продукта в соответствии с точным процессом приготовления. Это гарантирует, что необходимые ингредиенты будут смешаны в правильной пропорции и концентрациях. Для создания такого цифрового двойника используется SIMATIC IT R&D Suite, который поддерживает процесс разработки с учетом правил на различных рынках. Рецептура напитка определяется на основе одобренных ингредиентов. Дальше происходит моделирование и оптимизация в соответствии с требуемыми свойствами, например, количеством калорий. Процесс формирования продукта приводит к определению рецепта и соответствующей таблицы питательных веществ.

Далее происходит создание дизайна упаковки в виртуальное среде, например, с помощью SIEMENS NX. В результате получается цифровой двойник упаковки, содержащий данные
о компонентах и материалах, из которых она будет изготавливаться, физических свойствах, форме и размерах деталей, а также их 3D модели. Затем происходит моделирование стресс-теста цифрового двойника, например, с помощью Simcenter. С помощью моделирования мы можем, например, гарантировать, что жидкость не будет выливаться каждый раз, когда потребитель берёт открытую бутылку.

После создания цифрового двойника продукта можно перейти к разработке цифрового двойника процесса производства или технологии. Эта задача решается с помощью программного продукта COMOS, в котором планируется и проверятся производственный процесс, начиная с определения технологической схемы процесса, создания схемы трубопроводов и контрольно-измерительных приборов.

На основе цифрового двойника процесса производства можно построить цифровой двойник производственной линии. С помощью Tecnomatix Line Designer проектируется и при необходимости визуализируется компоновка производственных линий. Это цифровой двойник производственной линии, который поможет ещё до строительства реального объекта оптимизировать весь процесс на каждом шаге производства, вплоть до управления каждым устройством линии. При этом есть возможность использования цифровых двойников компонен-

тов и агрегатов линии от поставщиков и производителей оборудования.

При наличии цифрового двойника процесса производства и производственной линии можно, например, с помощью Tecnomatix Plant Simulation, оптимизировать потоки материала и пропускную способность, обнаружить узкие места и оптимизировать энергопотребление будущего завода.

При наличии цифрового двойника производства можно сделать виртуальную прогулку по этому производству, что очень хорошо подходит для обучения будущего персонала и проверки комфортности рабочей среды.

На основе имеющихся цифровых двойников оборудования, электрических схем и процесса производства, полученных на предыдущих этапах, можно создать в автоматическом режиме в TIA Portal программу, управляющую производством, для программируемых логических контроллеров.

Общий результат комплексно проверяется с помощью виртуального ввода в эксплуатацию в процессе моделирования завода, например, в Tecnomatix и на виртуальном контроллере SIMATIC на PLCSIM Advanced и виртуальном двойнике HMI. Такая взаимная интеграция позволяет произвести полное тестирование производства в виртуальной среде за счёт цифровых двойников.

Для достижения высокого уровня эффективности важно оптимизировать созданную последовательность производства, например, с помощью программного обеспечения SIMATIC Preactor. Это позволит планировать правильную и эффективную последовательность операций в соответствии с наличием ресурсов завода и возможными ограничениями.

Контроль качества продукции также закладывается на этапе создания производства. Например, можно использовать ПО SIMATIC IT Unified Architecture, позволяющее построить систему менеджмента качества, которая будет гарантировать качество продукции. Все возможные, даже минимальные отклонения на этапе производства немедленно будут поступать в систему. Это позволит обеспечить контроль качества и максимально снизить любые отклонения.

С помощью SIMATIC IT Unified Architecture Manufacturing Intelligence можно получить оценку производительности производственной линии. Также можно обеспечить глубокую детализацию временных измерений (например, вплоть до дня или смены), оборудования, установленного на каждом участке (вплоть до отдельных единиц), а также других параметров, связанных с заказами и продуктами.

2.8. Цифровые двойники в медицине

Как технология цифровых двойников используется в здравоохранении?

В системах здравоохранения цифровые двойники используются для создания цифровых представлений медицинских данных, таких как больничная среда, результаты лабораторных исследований, физиология человека и т. д., с помощью компьютерных моделей [16].

Технологию цифровых двойников можно использовать для создания виртуального двойника больницы для анализа операционных стратегий, возможностей, кадрового обеспечения и моделей ухода для выявления областей, требующих улучшения, прогнозирования возможных проблем и оптимизации организационных стратегий. Таким образом, цифровые двойники больниц позволяют:

- оптимизировать ресурсы. Использование исторических данных и данных в режиме реального времени о работе больниц и окружении (например, о случаях COVID-19, автомобильных авариях и т. д.) для создания цифровых двойников позволяет руководству больницы выявлять нехватку коек и оптимизировать графики работы персонала. Такая информация повышает эффективность использования ресурсов и оптимизирует работу больницы и персонала при одновременном снижении затрат;

- управлять рисками. Цифровые двойники обеспечивают безопасную среду для тестирования изменений в производительности системы (количества персонала, вакансий в операционных, обслуживания устройств и т. д.), что позволяет принимать стратегические решения на основе данных в сложной и чувствительной среде.

IBM Process Mining создает цифрового двойника организации, который можно применять для создания цифровых копий организационных процессов в медицинских учреждениях. Это позволяет запускать симуляции, выявлять узкие места до их возникновения и выявлять возможности автоматизации.

Цифровые двойники также применяются для моделирования органов и отдельных клеток или индивидуального генетического состава, физиологических характеристик и привычек для создания персонализированных лекарств и планов лечения. Эти копии внутренних систем человеческого тела улучшают медицинское обслуживание и лечение пациентов за счет индивидуальных диагностики и планирования лечения.

Цифровые двойники могут улучшить проектирование, разработку, тестирование и мониторинг новых лекарств и медицинских устройств.

Например, цифровые двойники лекарств и химических веществ позволяют модифицировать или переделывать лекарства с учетом размера частиц и характеристик состава для повышения эффективности. Цифровые двойники медицинского устройства позволяют разработчикам тестировать характеристики или использование устройства, вносить изменения в дизайн или материалы, а также проверять успех или неудачу модификаций в виртуальной среде перед производством. Это значительно снижает число отказов и повышает производительность и безопасность конечного продукта.

Каковы проблемы цифровых двойников в здравоохранении?

Некоторые из проблем, с которыми сталкивается внедрение цифровых двойников в здравоохранении, включают:

- ограниченное принятие. Технология цифровых двойников не получила широкого распространения в клинической практике. С другой стороны, даже несмотря на то, что система здравоохранения наращивает использование цифровых двойников, утверждается, что это остается дорогим и доступным не для всех. Технология цифровых двойников станет преимуществом, предназначенным для людей с более высокими финансовыми возможностями, что приведет к неравенству в системе здравоохранения;

- качество данных. Система искусственного интеллекта в цифровых двойниках учится на доступных биомедицинских данных, но качество данных может оказаться плохим. Следовательно, анализ и представление таких данных становится проблематичным. Это в конечном итоге негативно влияет на модели, что также влияет на надежность моделей в процессах диагностики и лечения;

- конфиденциальность данных. Применение цифровых двойников требует от организаций здравоохранения и страховых компаний сбора все большего количества данных на индивидуальном уровне. Со временем эти организации здравоохранения получают подробный слепок информации о биологическом, генетическом, физическом состоянии и образе жизни человека.

2.9. Ditto

Eclipse Ditto — это инструмент для создания цифровых двойников для интернета вещей с открытым исходным кодом. Технология потенциально отражает большое количество цифровых двойников, живущих в цифровом мире с физическими объектами. Это упрощает разработку решений для разработчиков программного обеспечения, поскольку им не нужно знать, как и где именно связаны физические объекты. С помощью Ditto объект можно использовать как любой другой веб-сервис через его цифрового двойника [17].

Ditto не является полноценной платформой интернета вещей. Он не предоставляет программное обеспечение, работающее на шлюзах интернета вещей, и не определяет и не реализует протокол интернета вещей для связи с устройствами. Основное внимание уделяется внутренним сценариям через веб-API для упрощения работы с уже подключенными устройствами и объектами из клиентских приложений или другого серверного программного обеспечения. Ditto также не указывает, какие данные или какую структуру должен предоставлять объект в интернете вещей.

Ditto можно использовать, чтобы получить полноценный API с поддержкой авторизации (HTTP, WebSocket и другие протоколы обмена сообщениями) для взаимодействия с цифровыми двойниками.

В типичном сценарии использования интернета вещей есть несколько мест для реализации программного обеспечения:

- на оборудовании или рядом с ним с использованием языков программирования высокого уровня C/C++ или Python;

- опционально на шлюзе, устанавливающем подключение к Интернету (например, на основе Eclipse Kura);

- в мобильном или веб-приложении с использованием языков программирования высокого уровня Java, Javascript, Swift и т. д.;

- в серверной части, выполняющей задачи предоставления прикладного интерфейса (API), маршрутизации запросов, обеспечения санкционированного доступа и предоставления данных и уведомлений.

Ditto фокусируется на решении серверных задач («бэкенда») в таких сценариях. Его цель — избавить решения интернета вещей от необходимости внедрять и эксплуатировать настраиваемую серверную часть. Вместо этого, используя Eclipse Ditto, пользователи могут сосредоточиться на бизнес-требованиях, на подключении устройств к облаку или серверной части и на внедрении бизнес-приложений.

2.10. iModel.js

iModel.js — это платформа с открытым исходным кодом для создания, использования и интеграции цифровых двойников инфраструктуры — цифровых представлений актива или системы, а также контекста и элементов управления. Операторы инфраструктуры используют цифровых двойников для лучшего планирования, эксплуатации и обслуживания активов [18].

Сердцем цифрового двойника инфраструк-

туры является реляционная база данных, известная как iModel, которая содержит компоненты, собранные из многих источников. Изменения в iModel управляются iModelHub и синхронизируются с распределенными копиями, создавая распределенную базу данных. Платформу iModel.js можно использовать для интеграции цифрового двойника инфраструктуры в цифровые рабочие процессы. Она содержит инструменты для создания, визуализации, запроса, анализа данных, синхронизации, согласования и защиты цифрового двойника.

iModel.js благодаря гибкости и открытости легко использовать и интегрировать с другими системами. В нем используются стандартные облачные и веб-технологии, выбранные для снижения входного сопротивления существующей кодовой базы и сохранения гибкости с течением времени.

3. Математическое моделирование

3.1. Математические модели

Модели можно поделить на детерминированные и стохастические. В детерминированных моделях все факторы, оказывающие влияние на развитие ситуации принятия решения, однозначно определены и их значения известны в момент принятия решения. Стохастические модели предполагают наличие элемента неопределенности, учитывают возможное вероятностное распределение значений факторов и параметров, определяющих развитие ситуации [19].

Среди детерминированных моделей широко распространены задачи линейного программирования. Линейное программирование — это набор методов, используемых в математическом программировании, также называемых математической оптимизацией. Эти методы используются для решения систем линейных уравнений и неравенств, перед которыми стоит цель максимизации или минимизации некоторой линейной функции. Если переменные решения должны быть целыми числами, такая модель называется целочисленной моделью программирования.

Пример ситуаций, в которых может быть полезна модель детерминированной оптимизации:

- ассортимент продукции — определить, какое количество продукции каждого типа необходимо производить с учетом ограничений на имеющиеся ресурсы;

- планирование производства — определить, сколько каждого вида продукции нужно производить в разные периоды времени, чтобы выполнить заданные объемы производства к определенному времени;

- смешивание — определить наилучшую смесь исходных материалов для минимизации затрат на производство смеси;

- резка материала — определить лучший способ резки ресурсного материала, чтобы максимизировать прибыль. Например, бревно можно распилить на пиломатериалы разных размеров, которые можно продать по разным ценам;

- кадровое обеспечение — определить наилучший способ назначения людей на работу, чтобы максимизировать их предпочтения или максимизировать производительность, исходя из их способностей на разных работах;

- транспортировка — определить наилучший способ маршрутизации ресурсов через транспортную сеть, чтобы минимизировать затраты, при этом доставив соответствующее количество ресурсов в каждое место;

- назначение — определить наилучший способ назначения ресурсов задачам;

- задача коммивояжера — определить лучший маршрут среди множества точек, который содержит каждую точку хотя бы один раз.

3.2. Excel Solver

Solver — это плагин для Microsoft Excel для решения задач оптимизационного анализа [20].

Оптимизационный анализ представляет собой более сложное расширение целевого анализа. Вместо того, чтобы устанавливать конкретное целевое значение для переменной, цель состоит в том, чтобы найти оптимальное значение для одной или нескольких целевых переменных при определенных ограничениях. Затем одна или несколько других переменных многократно изменяются с учетом указанных ограничений, пока не будут найдены наилучшие значения для целевых переменных.

В Excel можно использовать Solver, чтобы найти оптимальное значение (максимальное, минимальное, или определенное значение) для формулы в одной ячейке, называемой целевой ячейкой, с учетом определенных ограничений или ограничений для значений других ячеек формулы на листе. Это означает, что Solver работает с группой ячеек, называемых переменными решения, которые используются при вычислении формул в ячейках целей и ограничений. Solver корректирует значения в ячейках переменных решения, чтобы удовлетворить ограничения на ячейки ограничений и получить результат, который вы хотите получить для целевой ячейки.

Solver можно использовать, чтобы найти оптимальные решения для различных проблем, таких как:

- определение ежемесячного ассортимента продукции для подразделения по производству

лекарств, которое максимизирует рентабельность;

- планирование рабочей силы в организации;
- решение транспортных проблем;
- финансовое планирование и бюджетирование.

Методы решения, используемые Solver:

- симплекс-метод. Используется для линейных задач;
- метод обобщенного приведенного градиента. Используется для нелинейных задач;
- эволюционный метод. Используется для нелинейных задач.

3.3. GIBBS

GIBBS — это компьютерная программа комплексного моделирования технологических процессов промысловой подготовки, переработки и транспорта природного и попутного газа, газового конденсата и нефти. Наиболее полный набор моделей позволяет описать свойства сырья и продуктов, определить затраты тепла, энергии, материально-тепловой баланс производства, массовые и объемные выходы и состав продуктов, их соответствие действующим стандартам [21].

Используемые теоретические методы, в основе которых лежат принципы современной технической термодинамики и использование уравнения состояния для расчета фазовых превращений и теплофизических свойств углеводородных смесей, являются универсальными и позволяют моделировать в широком диапазоне условий следующие технологические процессы:

- процессы промысловой подготовки природного газа, включая установки низкотемпературной сепарации и конденсации;
- процессы обработки газа с вводом, сбором и регенерацией ингибиторов гидратообразования;
- процессы промысловой и заводской подготовки и переработки газоконденсата и нефти, включая деэтанизацию, стабилизацию и фракционирование;
- процессы низкотемпературного выделения сжиженных углеводородных газов, этана, гелия и азота из природного газа, фракционирования смесей легких углеводородов, холодильные циклы, низкотемпературные детандерные заводы;
- процессы сжижения природного газа;
- процессы тепловых станций на водяном паре;
- процессы многофазного транспорта нефтегазоконденсатных смесей, включая транспорт неньютоновских жидкостей и водонефтяных эмульсий;
- процессы аминовой очистки газа от кислых компонентов.

Технологическая модель производства, построенная с помощью GIBBS, позволяет решить проблемы, возникающие на этапе его эксплуатации. Например, почему свойства продукта отличаются от ожидаемых, что делать, если изменилось качество сырья или его состав и т.п.

Пользуясь ПО GIBBS, инженер-технолог не просто работает с изображением технологической схемы. Он может создавать сложные многоуровневые модели производств, объединяющие несколько промыслов и комплексов переработки сырья. Каждый уровень может соответствовать установке, группе установок, заводу, крупному промысловому или заводскому комплексу, и все уровни объединяются в единую блок-схему. Кроме этого, каждый проект в GIBBS может иметь множество вариантов решений, отличающихся набором компонентов и технологическими параметрами.

GIBBS — вычислительная программа, разработанная для Windows-платформ, имеет простой интерфейс. Программа тесно интегрирована с Microsoft Office, но имеет и альтернативный генератор отчетов в виде привычных электронных таблиц. Вся информация системы положена на реляционную основу и для доступа к ней используется хорошо зарекомендовавший себя на задачах подобного класса процессор баз данных Borland Database Engine.

В итоге GIBBS — средство для повышения качества проектирования и эксплуатации нефтяных и газоконденсатных месторождений, определения оптимальных технологических условий промысловой обработки, транспортировки и заводской переработки углеводородного сырья.

3.4. FlexSim

FlexSim — это программный пакет для имитационного моделирования, разработанный FlexSim Software Products, Inc. В семейство продуктов FlexSim входит система общего назначения FlexSim и система моделирования для здравоохранения FlexSim HC [22].

FlexSim используется в различных проектах моделирования, включающих как стандартные, так и гибкие производственные системы. Примеры использования: оптимизация компонентов смеси при производстве кормов, планировании производственных линий, оптимизация сборочных линий электроники, планирование производства стали. FlexSim может использоваться для разработки компьютерных имитационных моделей для приложений Промышленности 4.0.

FlexSim можно расширить с помощью C++, что позволяет интегрировать программное обеспечение в системы, связанные с передачей данных в реальном времени. В одном исследовании FlexSim был интегрирован в динамическую систему приложений, управляемую данными, для

автоматического создания имитационных моделей с помощью языка XML [23].

Стандартная библиотека объектов FlexSim содержит объект робота с 6 осями, способный использовать как предварительно созданную логику движения, так и настраивать пути движения. FlexSim использовался для моделирования и анализа роботизированных ячеек в производственных условиях, включая динамическое планирование и управление роботизированной сборочной ячейкой. Стандартная библиотека объектов также содержит объект крана, предназначенный для имитации кранов с рельсовым управлением, таких как козловые, мостовые или стреловые краны.

В апреле 2009 года компания FlexSim Software Products, Inc. выпустила автономный продукт для моделирования здравоохранения под названием FlexSim HC. Он был разработан как пакет моделирования, ориентированный на моделирование потоков пациентов и других процессов здравоохранения. На практике среда FlexSim HC используется организациями здравоохранения для оценки различных сценариев в их процессах и проверки сценариев перед реализацией. Среда использовалась в различных инициативах по улучшению ухода за пациентами, в том числе в исследованиях для понимания различных вариантов лечения в родовспоможении и привлечения опытных медсестер для лечения несрочных пациентов. Во время пандемии COVID-19 FlexSim HC использовался для анализа усилий по развертыванию вакцинации и улучшения потока пациентов в пунктах вакцинации [24].

В качестве программного обеспечения для моделирования общего назначения FlexSim используется в ряде областей:

- погрузочно-разгрузочные работы: конвейерные системы, упаковка, складирование;
- логистика и распределение: работа контейнерного терминала, проектирование цепочки поставок, рабочий процесс распределительного центра, схема обслуживания и хранения и т. д.;
- транспорт: транспортный поток на автомагистралях, пешеходный поток на транзитных станциях, координация морских судов, прохождение таможни и т. д.;
- прочее: процессы добычи полезных ископаемых, сетевое взаимодействие потоков данных и т. д.

FlexSim включает в себя стандартную библиотеку объектов, каждый из которых содержит предварительно созданную логику и может выполнять задачи для имитации ресурсов, используемых в реальных операциях. Объекты FlexSim делятся на 4 класса: класс фиксированных ресурсов, класс исполнителя задач, класс узлов и класс визуальных объектов. FlexSim использует объектно-ориентированный дизайн.

Логика модели FlexSim может быть построена с использованием очень небольшого количества компьютерного кода или вообще без него. Большинство стандартных объектов содержат набор раскрывающихся списков, окон свойств и триггеров, которые позволяют пользователю настраивать логику, необходимую для точной модели системы. FlexSim также включает инструмент построения блок-схем для создания логики модели с использованием предварительно созданных блоков действий.

Пользователи могут создавать модели, перетаскивая предопределенные 3D-объекты в «вид модели», чтобы компоновать и связывать модель. Опытные пользователи также могут задавать и изменять параметры и поведение объектов с помощью языков программирования FlexScript и C++.

3.5. MATLAB

MATLAB (сокращение от англ. «Matrix Laboratory») — пакет прикладных программ для решения задач технических вычислений. Пакет функционирует на большинстве современных операционных систем [25].

Язык MATLAB является высокоуровневым интерпретируемым языком программирования, включающим основанные на матрицах структуры данных, широкий спектр функций, объектно-ориентированные возможности и интерфейсы к программам, написанным на других языках программирования. Основной особенностью языка MATLAB являются его широкие возможности по работе с матрицами.

Программы, написанные на MATLAB, бывают двух типов — функции и скрипты. Функции имеют входные и выходные аргументы, а также собственное рабочее пространство для хранения промежуточных результатов вычислений и переменных. Скрипты же используют общее рабочее пространство. Как скрипты, так и функции сохраняются в виде текстовых файлов и динамически компилируются в машинный код. Существует также возможность сохранять т.н. pre-parsed программы — функции и скрипты, обработанные в удобный для машинного исполнения вид. В общем случае такие программы выполняются быстрее обычных.

Пакет MATLAB включает различные интерфейсы (т.н. MATLAB API) для получения доступа к внешним подпрограммам, написанным на других языках программирования, данным, клиентам и серверам, общающимся через технологии Component Object Model или Dynamic Data Exchange, а также периферийным устройствам, которые взаимодействуют напрямую

с MATLAB.

Для MATLAB имеется возможность создавать специальные наборы инструментов, расширяющие его функциональность. Наборы инструментов представляют собой коллекции функций и объектов, написанных на языке MATLAB для решения определённого класса задач. Компания Mathworks поставляет наборы инструментов, которые используются во многих областях, включая следующие:

- цифровая обработка сигналов, изображений и данных: Signal Processing Toolbox, DSP System Toolbox, Image Processing Toolbox, Wavelet Toolbox, Communications System Toolbox — наборы функций и объектов, позволяющих решать широкий спектр задач обработки сигналов, изображений, проектирования цифровых фильтров и систем связи;

- системы управления: Control Systems Toolbox, Robust Control Toolbox, System Identification Toolbox, Model Predictive Control Toolbox, Model-Based Calibration Toolbox — наборы функций и объектов, облегчающих анализ и синтез динамических систем, проектирование, моделирование и идентификацию систем управления, включая современные алгоритмы управления, такие как робастное управление, H∞-управление, ЛМН-синтез, μ-синтез и другие;

- финансовый анализ: Econometrics Toolbox, Financial Instruments Toolbox, Financial Toolbox, Datafeed Toolbox, Trading Toolbox — наборы функций и объектов, позволяющие быстро и эффективно собирать, обрабатывать и передавать различную финансовую информацию;

- анализ и синтез географических карт, включая трёхмерные: Mapping Toolbox;

- сбор и анализ экспериментальных данных: Data Acquisition Toolbox, Image Acquisition Toolbox, Instrument Control Toolbox, OPC Toolbox — наборы функций и объектов, позволяющих сохранять и обрабатывать данные, полученные в ходе экспериментов, в том числе в реальном времени. Поддерживается широкий спектр научного и инженерного измерительного оборудования;

- визуализация и представление данных: Virtual Reality Toolbox — позволяет создавать интерактивные миры и визуализировать научную информацию с помощью технологий виртуальной реальности и языка VRML;

- средства разработки: MATLAB Builder for COM, MATLAB Builder for Excel, MATLAB Builder for NET, MATLAB Compiler, HDL Coder — инструменты, позволяющие создавать независимые приложения из среды MATLAB;

- взаимодействие с внешними программными продуктами: MATLAB Report Generator, Excel Link, Database Toolbox, MATLAB Web Server, Link for ModelSim — наборы функций, позволяющие сохранять данные различных видов таким образом, чтобы другие программы могли с ними работать;

- базы данных: Database Toolbox — инструменты работы с базами данных;

- научные и математические пакеты: Bioinformatics Toolbox, Curve Fitting Toolbox, Fixed-Point Toolbox, Optimization Toolbox, Global Optimization Toolbox, Partial Differential Equation Toolbox, Statistics And Machine Learning Toolbox, RF Toolbox — наборы специализированных математических функций и объектов, позволяющие решать широкий спектр научных и инженерных задач, включая разработку генетических алгоритмов, решения задач в частных производных, целочисленные проблемы, оптимизацию систем и другие;

- нейронные сети: Neural Network Toolbox — инструменты для синтеза и анализа нейронных сетей;

- нечёткая логика: Fuzzy Logic Toolbox — инструменты для построения и анализа нечётких множеств;

- символьные вычисления: Symbolic Math Toolbox [26] — инструменты для символьных вычислений с возможностью взаимодействия с символьным процессором программы Maple.

Помимо вышеперечисленных, существуют другие наборы инструментов для MATLAB, написанные сторонними компаниями и энтузиастами.

3.6. Simulink

Simulink — среда динамического моделирования сложных технических систем и инструмент для модельно-ориентированного проектирования. Его основным интерфейсом является графический инструмент для построения диаграмм и настраиваемый набор библиотек блоков. Он предлагает тесную интеграцию со средой MATLAB и может либо использовать MATLAB, либо создавать сценарии из него. Simulink широко используется в автоматическом управлении и цифровой обработке сигналов для многодоменного моделирования и проектирования на основе моделей. В сочетании с другими продуктами Simulink может автоматически генерировать исходный код на языке C для реализации систем в режиме реального времени [27].

Simulink предназначен для моделирования и симуляции на системном уровне, что позволяет проводить всестороннее исследование разрабатываемой системы в единой среде проектирования. Моделирование и симуляции позволяют провести проверку поведения системы в критических условиях или аварийных сценариях.

Тем самым происходит снижение затрат на дорогостоящие физические прототипы. Проверка системы осуществляется с помощью полунатурного моделирования и быстрого прототипирования.

Модели Simulink поддерживают автоматическую генерацию кода промышленного качества на языках C, C++ и HDL [28]. Результаты работы сгенерированного кода и модели идентичны. Следующим шагом является развертывание кода на целевом вычислителе или FPGA/ASIC.

Simulink использует преимущества MATLAB. Алгоритмы, созданные в MATLAB, не требуется переделывать для повторного использования в Simulink. Код MATLAB добавляется в блок Simulink или диаграмму Stateflow без изменений. Так же MATLAB используется для создания наборов входных данных для симуляций систем. Симуляции масштабируются на кластеры для ускорения таких инженерных задач как перебор параметров или оптимизация, а затем их результаты анализируются и визуализируются в MATLAB.

3.7. GNU Octave

GNU Octave — свободная программная система с открытым исходным кодом для математических вычислений, использующая совместимый с MATLAB язык высокого уровня [29].

Octave представляет интерактивный командный интерфейс для решения линейных и нелинейных математических задач, а также проведения других численных экспериментов. Кроме того, Octave можно использовать для пакетной обработки. Язык Octave оперирует арифметикой вещественных и комплексных скаляров, векторов и матриц, имеет расширения для решения линейных алгебраических задач, нахождения корней систем нелинейных алгебраических уравнений, работы с полиномами, решения различных дифференциальных уравнений, интегрирования систем дифференциальных и дифференциально-алгебраических уравнений первого порядка, интегрирования функций на конечных и бесконечных интервалах. Этот список можно расширить, используя язык Octave (или используя динамически загружаемые модули, созданные на языках C, C++, Фортран и др.).

3.8. AnyLogic

AnyLogic — инструмент имитационного моделирования для бизнеса для получения детального представления о бизнес-системах и процессах и их оптимизации [30].

AnyLogic позволяет хранить модели в облаке и интегрировать их с оперативными данными, проводить эксперименты и делиться результатами с конечными пользователями, чтобы они могли сразу применять результаты моделирования для решения текущих задач. Модель в AnyLogic Cloud можно интегрировать в рабочие процессы и использовать для создания цифровых двойников. Для того, чтобы поделиться моделями с клиентами используется готовая облачная среда, благодаря чему для их запуска не нужно устанавливать дополнительное ПО. В AnyLogic Cloud можно управлять доступом к моделям, обновлять их и создавать собственные интерфейсы для моделей. Пользователи могут настроить эксперименты с помощью интерактивных панелей управления, посмотреть анимацию и получить результаты проведённых экспериментов из общей базы данных.

В AnyLogic Cloud можно систематизировать и редактировать данные экспериментов и сценариев, включая версии моделей, параметры запусков и результаты моделирования. Сервис также позволяет визуально сравнивать результаты экспериментов, в том числе разницу ключевых показателей в разных прогонах.

В облачной системе многопрогонные эксперименты выполняются быстрее и эффективнее, чем на обычном персональном компьютере. Сервис оперативно реагирует на потребности в вычислительных мощностях, подключая дополнительные узлы и ядра для выполнения экспериментов. Все эксперименты вместе со входными данными и результатами хранятся в облаке. Если входные данные у запусков одинаковы, AnyLogic Cloud использует предыдущие результаты, ускоряя новый эксперимент и экономя вычислительные ресурсы.

Для визуализации результатов экспериментов в AnyLogic Cloud есть графики и диаграммы, включая продвинутые — например, диаграмма размаха, точечная диаграмма и диаграмма поверхности.

4. Заключение

Развитие информационных технологий привело к использованию новых методов, таких как создание цифровых двойников или использование искусственного интеллекта. Изучение возможностей применения этих методов является одним из направлений исследований.

Использование мощных современных программных средств моделирования, включающих инструменты решения уравнений различного типа, позволяет существенно упростить разработку программных средств автоматизации технологических процессов. Создание моделей вместо использования реальных промышленных объектов снижает трудозатраты, стоимость и безопасность разработки.

В вышеприведенном обзоре рассмотрены как

классические, так и новые методы моделирования в разных видах деятельности, включая производство, медицину и бизнес-процессы. Описано использование различных математических пакетов и систем создания цифровых двойников. Отдельно отмечены проекты с открытым исходным кодом.

Существуют и другие программные продукты для решения различных задач моделирования, однако, они выходят за рамки данного обзора.

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

Overview of existing solutions in the field of technological process modeling

Yaroslav Zotov

Abstract. The article contains an overview of some existing modeling solutions in some industries. Definitions of digital twins are given. Descriptions of programs implementing the functionality of digital twins, including open source solutions, are given. Some mathematical solutions are described.

Keywords: modeling, digital twin, automated control systems

Литература

1. Aidan Fuller, Zhong Fan, Charles Day. Digital Twin. Enabling Technologies, Challenges and Open Research. IEEE Access, 2020.

2. E. Glaessgen, D. Stargel. The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles. 53rd AIAA Structures, Structural D ynamics and Materials Conference, American Institute of Aeronautics and Astronautics, Honolulu, Hawaii, 2012.

3. Y. Chen. Integrated and Intelligent Manufacturing: Perspectives and Enablers. Engineering, vol. 3, pp. 588-595, 2017.

4. Z. Liu, N. Meyendorf, N. Mrad. The role of data fusion in predictive maintenance using digital twin. Annual Review Of Progress In Quantitative Nondestructive Evaluation, Provo, Utah, USA, 2018.

5. Y. Zheng, S. Yang, and H. Cheng. An application framework of digital twin and its case study. Journal of Ambient Intelligence and Humanized Computing, vol. 10, pp. 1141-1153, 2018.

6. R. Vrabi˘c, J. A. Erkoyuncu, P. Butala, R. Roy. Digital twins: Understanding the added value of integrated models for through-life engineering services. Procedia Manufacturing, vol. 16, pp. 139-146, 2018.

7. A. Madni, C. Madni, S. Lucero. Leveraging Digital Twin Technology in Model-Based Systems Engineering. Systems, vol. 7, p. 7, 2019.

8. ГОСТ Р 57700.37–2021. Компьютерные модели и моделирование. Цифровые двойники изделий. Общие положения. 2021.

9. GE Predix [Электронный ресурс] // TAdviser. Режим доступа: https://www.tadviser.ru/index.php/%D0%9F%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82:GE_Predix (Дата обращения: 12.08.2022).

10. Thingworx. The power of now [Электронный ресурс] // PTC Software. Режим доступа: https://www.ptc.com/-/media/Files/PDFs/ThingWorx/ThingWorx-Overview-Brochure-2.pdf (Дата обращения: 10.08.2022).

11. Thingworx.Промышленный интернет вещей [Электронный ресурс] // Ирисофт. Режим доступа: https://www.irisoft.ru/products/thingworx/ (Дата обращения: 12.08.2022).

12. Анонс комплекса для создания цифровых двойников дорожных объектов [Электронный ресурс] // TAdviser. Режим доступа: https://www.tadviser.ru/index.php/%D0%9F%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82:%D0%90%D0%B2%D1%82%D0%BE%D0%B4%D0%B8%D1%81%D0%BA%D0%B0%D0%B2%D0%B5%D1%80%D0%B8 (Дата обращения: 12.08.2022).

13. Zenkit – Your Digital Twin [Электронный ресурс] // Zenkit Suite. Режим доступа: https://zenkit.com/en/suite/solutions/digital-twin/ (Дата обращения: 12.08.2022).

14. Ansys Twin Builder [Электронный ресурс] // Моделирование и цифровые двойники. Режим доступа: https://www.cadfem-cis.ru/products/ansys/systems/twinbuilder/ (Дата обращения: 20.08.2022).

15. Применение цифрового двойника на примере производства напитков [Электронный ресурс] // Промэнерго Автоматика. Режим доступа: https://www.siemens-pro.ru/articles/siemens-articles-111.html (Дата обращения: 20.08.2022).

16. Hazal Şimşek. 3 Best Use Cases of Digital Twin in Healthcare & Their Benefits [Электронный ресурс] // AIMultiple. Режим доступа: https://research.aimultiple.com/digital-twin-healthcare/ (Дата обращения: 05.09.2022).

17. Eclipse Ditto documentation overview [Электронный ресурс] // Eclipse Foundation. Режим доступа: https://www.eclipse.org/ditto/intro-overview.html (Дата обращения: 05.09.2022).

18. About iModel.js [Электронный ресурс] // iTwin.js. Режим доступа: https://www.itwinjs.org/v1/ (Дата обращения: 05.09.2022).

19. Boardman. Introduction to Industrial Engineering. Arlington, Mavs Open Press, 2020.

20. Optimization with Excel Solver [Электронный ресурс] // Simply Easy Learning at your fingertips. Режим доступа: https://www.tutorialspoint.com/excel_data_analysis/advanced_data_analysis_optimization_with_excel_solver.htm (Дата обращения: 20.09.2022).

21. Что такое GIBBS [Электронный ресурс] // GIBBS. Моделирование в нефтегазовой отрасли. Режим доступа: http://www.gibbsim.ru/node/10 (Дата обращения: 20.09.2022).

22. FlexSim The most powerful, capable, and easy-to-use 3D simulation software [Электронный ресурс] // 3D Simulation Modeling and Analysis Software. Режим доступа: https://www.flexsim.com/flexsim/ (Дата обращения: 20.09.2022).

23. Krenczyk Damian. Data-driven modelling and simulation for integration of production planning and simulation systems. Selected Engi-neering Problems (3): 119–122, 2021.

24. Windsor Matt. Simulations help vaccine rollout and cut the wait in UAB clinics. UAB Reporter. The University of Alabama at Birming-ham. 2021.

25. MATLAB [Электронный ресурс] // MathWorks. Режим доступа: https://www.mathworks.com/products/matlab.html (Дата обращения: 04.10.2022).

26. A Brief History of MATLAB [Электронный ресурс] // MathWorks. Режим доступа: https://www.mathworks.com/company/newsletters/articles/a-brief-history-of-matlab.html (Дата обращения: 05.10.2022).

27. Simulink [Электронный ресурс] // Exponenta. Режим доступа: https://exponenta.ru/simulink (Дата обращения: 05.10.2022).

28. Simulink – Simulation and Model-Based Design [Электронный ресурс] // MathWorks. Режим доступа: https://www.mathworks.com/products/simulink.html (Дата обращения: 05.10.2022).

29. GNU Octave [Электронный ресурс]. Режим доступа: https://octave.org/ (Дата обращения: 05.10.2022).

30. AnyLogic: имитационное моделирование для бизнеса [Электронный ресурс]. Режим доступа: https://www.anylogic.ru/ (Дата обращения: 05.10.2022).

Машинно-ориентированный текстовый интерфейс отладчика GDB

В.А. Галатенко[1], К.А. Костюхин[2]

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

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

Аннотация. Интерактивный отладчик GDB является основным средством отладки программ, разрабатываемых на инструментальной платформе Linux. Предоставляя широкие возможности для отладки, GDB, тем не менее, не способен удовлетворить всех потребностей разработчиков, например, многопроцессорных систем. Одним из решений является рассматриваемый в этой статье текстовый интерфейс GDB/MI, посредством которого сторонние разработчики могут расширять функциональность отладчика.

Ключевые слова: отладка, многопроцессорные системы, текстовый интерфейс, графический интерфейс, GDB, MI.

1. Введение

Разработчики многопроцессорных систем нередко сталкиваются с проблемой интерактивной отладки как системы в целом, так и отдельных ее компонентов. Учитывая, что отладчик GDB [1], де-факто, является стандартным отладчиком для unix-подобных систем, то выглядит вполне естественным попытаться использовать именно его.

Авторы видят две возможности развития этой идеи: предоставление пользователю графического интерфейса, использующего запущенные экземпляры GDB по одному, как минимум, на каждый узел многопроцессорной системы, и модификация самого GDB для поддержки многопроцессорной отладки. Второй способ является более трудоемким и требует поддержки со стороны сообщества Free Software Foundation, поэтому в данной работе мы сосредоточимся на первом способе. Ключевым моментом здесь является машинно-ориентированный текстовый интерфейс GDB/MI [2]. Фраза «машинно-ориентированный» говорит о том, что этот интерфейс создан для обеспечения коммуникации между отладчиком GDB и другими программами, например, так называемыми графическими надстройками (GUI frontends).

Идеи о создании графических надстроек для GDB, который в своем арсенале имеет только интерфейс командной строки (Command Line Interface, CLI), витали в воздухе уже давно. До того, как появился MI, программы типа DDD (Data Display Debugger [3]), предоставляющие графический отладочный интерфейс разработчикам, использовали стандартные ввод и вывод процесса GDB для того, чтобы, имитируя пользователя, подавать CLI-команды ему на вход и перехватывать с последующей обработкой результат выполнения команд на выходе. У этого подхода есть один существенный недостаток. При изменении, даже незначительном, формата вывода результата выполнения команды графическая надстройка оказывается не в состоянии правильно интерпретировать результат.

Поэтому, начиная с версии 5, в GDB появился дополнительный интерфейс GDB/MI (далее просто MI), основным преимуществом которого является его независимость от формата отображения результата выполнения команды GDB. GDB/MI (расшифровывается как GDB Machine Interface) представляет собой набор структурированных строк, из которых состоят как запросы к GDB от внешних программ, так и их результаты выполнения.

2. Общее описание MI

Перед началом сеанса отладки необходимо запустить GDB с аргументом

--interpreter=mi

Означающим, что все данные, поступающие на стандартный ввод, GDB будет интерпретировать, как команды MI. Опционально можно указать номер версии MI (mi1, mi2 или mi3). По умолчанию используется последняя версия интерфейса.

Взаимодействие внешней программы, использующей MI, с отладчиком GDB состоит из трех основных частей:

- команды, отправленные в GDB (входные команды);
- результаты выполнения этих команд;
- оповещения от GDB о произошедших в отлаживаемой программе или в самом отладчике событиях.

Информационные команды, которые не влияют на статус выполнения отлаживаемой программы, получают в качестве результата всю требуемую информацию. Команды, которые запускают отлаживаемую программу на выполнение, получают только результат: успешно запущена или нет. Оповещения могут показывать изменившийся статус отлаживаемой программы: остановлена, завершена, произошла смена текущего потока управления.

Полный перечень всех команд MI приведен в [2]. В этой работе мы ограничимся общим описанием синтаксиса команд, а также базовых команд, наиболее важных для организации процесса отладки.

3. Синтаксис команд MI

3.1. Входные команды

В общем случае синтаксис входных команд MI выглядит следующим образом.

Команда -> CLI-команда | MI-команда
CLI-команда -> [*токен*] *<команда GDB>*
MI-команда ->[*токен*]-*команда* [-*параметр1*] [*значение1*] … [-*параметрN*] [*значениеN*] [*аргумент1*] … [*аргументN*] [-- *опция*]

Где

токен – [0..9]*, опциональная последовательность десятичных цифр (если токен указан, то результат выполнения команды также будет его содержать).
<команда GDB> - обычная команда отладчика GDB.
команда – одна из команд MI, описанных ниже, а также в [2].
параметр1…параметрN – опциональный набор параметров команды.
значение1…значениеN – опциональный набор значений параметров.
аргумент1…аргументN – опциональный набор аргументов команды.
опция – дополнительная опция команды.

3.2. Ответ отладчика

Ответ отладчика состоит из нескольких записей приведенного в этом разделе формата. Последовательность записей всегда завершается строкой *(gdb)*.

Ответ -> информационная запись | результат (gdb)

Где

результат -> [*токен*]^*статус*[,*данные*]
информационная запись -> асинхронный ответ | дополнительная информация
статус -> "done"| "running"| "connected"| "error"| "exit"
данные -> переменная=значение
переменная – имя внутренней переменной, содержащей информацию по запросу.
значение – набор сгруппированных данных.
асинхронный ответ - *<результат выполнения отлаживаемой программы до останова или завершения> | =<дополнительное сообщение отладчика> | +<статус целевой программы>.
дополнительная информация - ~<вывод консоли отладчика> | @<вывод целевой программы> | &<вывод журнала>.

3.3. Пример

Пример ниже иллюстрирует приведенное описание.

```
$ gdb -q --interpreter=mi ./example
=thread-group-added,id="i1"
~"Reading symbols from ./example..."
~"done.\n"
(gdb)
-break-insert main
^done,bkpt={number="1",type="breakpoint",
disp="keep",enabled="y",addr="0x08048457",
func="main",file="example.c", fullname="/opt/ex/ex-
ample.c",line="10", thread-
groups=["i1"],times="0",original-location="main"}
(gdb)
-exec-run
=thread-group-started,id="i1",pid="2956"
=thread-created,id="1",group-id="i1"
^running
*running,thread-id="all"
(gdb)
=breakpoint-modified,bkpt={number="1",
type="breakpoint",disp="keep",enabled="y",
addr="0x08048457",func="main", file="exam-
ple.c",fullname="/opt/ex/example.c", line="10",thread-
groups=["i1"],times="1",original-location="main"}
~"\n"
~"Breakpoint 1, main () at example.c:10\n"
~"10\t\tint i, k=9;\n"
*stopped,reason="breakpoint-
hit",disp="keep",bkptno="1",
frame={addr="0x08048457",func="main",args=[],file
="example.c",fullname="/opt/ex/example.c",
line="10"},thread-id="1",stopped-threads="all",core="0"
(gdb)
-exec-continue
^running
*running,thread-id="all"
(gdb)
~"[Inferior 1 (process 2956) exited with code 012]\n"
=thread-exited,id="1",group-id="i1"
=thread-group-exited,id="i1",exit-code="012"
*stopped,reason="exited",exit-code="012"
(gdb)
-gdb-exit
^exit
```

4. Основные команды MI

4.1. Работа с точками прерывания

-break-after – аналог GDB-команды *ignore*.

Игнорировать точку прерывания указанное количество ее срабатываний.

-break-commands – аналог GDB-команды *commands*. Задает список команд GDB, которые будут выполнены при срабатывании указанной точки прерывания.

-break-condition – аналог GDB-команды *condition*. Задает условия срабатывания точки прерывания.

-break-delete – аналог GDB-команды *delete*. Удаляет одну или несколько точек прерывания.

-break-disable – аналог GDB-команды *disable*. Временно отключает точку прерывания.

-break-enable – аналог GDB-команды *enable*. Включает ранее отключенную точку прерывания.

-break-insert – аналог GDB-команды *break, tbreak*. Устанавливает точку прерывания.

-break-list – аналог GDB-команды *info break*. Выдает информацию по установленным точкам прерывания.

-break-watch – аналог GDB-команды *watch*. Устанавливает точку прерывания по данным.

4.2. Работа с потоками управления

-thread-info – аналог GDB-команды *info thread*. Выдает информацию по отлаживаемым потокам управления.

-thread-select – аналог GDB-команды *thread*. Устанавливает заданный поток управления в качестве текущего (по умолчанию к нему будут применяться все последующие команды отладки).

4.3. Управление ходом выполнения программы

-exec-continue – аналог GDB-команды *continue*. Запускает отлаживаемый поток управления.

-exec-finish – аналог GDB-команды *finish*. Запускает отлаживаемый поток управления до выхода из текущего кадра стека.

-exec-interrupt – аналог GDB-команды *interrupt*. Останавливает выполнение программы.

-exec-jump – аналог GDB-команды *jump*. Запускает отлаживаемый поток управления с указанного адреса.

-exec-next – аналог GDB-команды *next*. Запускает отлаживаемый поток управления до перехода на другую строку исходного текста без остановки в вызываемых в процессе выполнения функциях.

-exec-next-instruction – аналог GDB-команды *nexti*. Запускает отлаживаемый поток управления до перехода на другую машинную команду без остановки в вызываемых в процессе выполнения функциях.

-exec-return – аналог GDB-команды *return*. Производит немедленный возврат из текущего кадра стека.

-exec-run – аналог GDB-команды *run*. Создает новый поток управления с указанной точкой входа.

-exec-step – аналог GDB-команды *step*. Запускает отлаживаемый поток управления до перехода на другую строку исходного текста с остановкой в вызываемых в процессе выполнения функциях.

-exec-step-instruction – аналог GDB-команды *stepi*. Запускает отлаживаемый поток управления до перехода на другую ассемблерную команду с остановкой в вызываемых в процессе выполнения функциях. Выполняет ровно одну машинную команду.

-exec-until – аналог GDB-команды *until*. Запускает отлаживаемый поток управления до достижения заданного адреса.

4.4. Работа со стеком

-stack-info-frame – аналог GDB-команды *info frame*. Выдает информацию о заданном кадре стека.

-stack-list-frames – аналог GDB-команды *backtrace*. Выдает информацию о стеке вызовов текущего потока управления.

-stack-list-locals – аналог GDB-команды *info locals*. Выдает информацию о локальных переменных текущего кадра стека.

-stack-select-frame – аналог GDB-команды *frame*. Делает заданный кадр стека текущим.

4.5. Работа с данными

-data-disassemble – аналог GDB-команды *disassemble*. Выдает дизассемблированный фрагмент памяти целевого потока.

-data-evaluate-expression – аналог GDB-команды *print* или *call*. Вычисляет выражение и выдает его результат.

-data-list-changed-registers – полезная команда для графических надстроек. Выдает список регистров, изменившихся с момента последнего останова отлаживаемого потока.

-data-list-register-names – выдает список имен доступных для отладочных действий регистров целевого процессора.

-data-list-register-values – аналог GDB-команды *info registers*. Выдает список целевых регистров с их значениями.

-data-read-memory – аналог GDB-команды *x*. Выдает содержимое целевой памяти в указанном формате и диапазоне адресов.

-data-read-memory-bytes – пытается прочесть и вернуть содержимое всех доступных ячеек памяти в указанном диапазоне адресов. Если содержимое памяти по каким-то адресам прочесть не удается, то соответствующие адреса помечаются как нечитаемые.

-data-write-memory-bytes – записывает массив

байтов в целевую память по указанному диапазону адресов.

4.6. Работа с таблицами имен

-symbol-info-functions – аналог GDB-команды *info functions*. Выводит список всех глобальных функций, удовлетворяющих заданному шаблону.

-symbol-info-types – аналог GDB-команды *info types*. Выводит список всех определенных в модуле типов, удовлетворяющих заданному шаблону.

-symbol-info-variables – аналог GDB-команды *info variables*. Выводит список всех глобальных переменных, удовлетворяющих заданному шаблону.

-symbol-list-lines – Выводит список всех строк текста программы из заданного в качестве аргумента исходного файла, содержащих исполняемый код.

4.7. Выбор целевой системы

-target-select – аналог GDB-команды *target*. Устанавливает для отладчика указанную целевую систему.

-target-disconnect – аналог GDB-команды *disconnect*. Завершает сеанс отладки текущей целевой системы.

4.8. Команды поддержки MI

-info-gdb-mi-command – возвращает true или false в зависимости от того, поддерживает ли указанную команду отладчик, или нет.

-list-features – возвращает список поддерживаемых отладчиком настроек и команд MI протокола.

-list-target-features – возвращает список расширенных свойств целевой системы, которые может использовать отладчик, например, поддерживает ли целевая система асинхронное выполнение отлаживаемых потоков, или обратное выполнение (reverse execution).

4.9. Вспомогательные команды MI

-gdb-exit – аналог GDB-команды *quit*. Производит завершение работы отладчика.

-gdb-set – аналог GDB-команды *set*. Устанавливает значение переменной отладчика или меняет его настройки.

-gdb-show – аналог GDB-команды *show*. Отображает значение переменной отладчика или его настроек.

-gdb-version – аналог GDB-команды *show version*. Отображает версию используемого отладчика.

-enable-timings – переключает режим отображения времени выполнения команды отладчика. Включенный режим позволяет разработчикам оптимизировать свой код, например, смотреть, как долго выполнялся фрагмент программы между двумя точками останова.

5. Недостатки интерфейса MI

Разработчикам, планирующим внедрять интерфейс MI в свои отладочные комплексы, следует иметь ввиду, что MI является именно интерфейсом, а не протоколом отладки. В частности, можно выделить следующие его недостатки.

Во-первых, отсутствует строгая спецификация MI. В официальной документации [2] есть ряд неточностей, с которыми сталкивались авторы. В частности, в описании команды *-exec-step-instruction* нет упоминания о том, что в качестве аргумента можно задавать число машинных команд, которые должны быть выполнены (по умолчанию выполняется только одна машинная команда).

Во-вторых, если вывод отлаживаемой программы направлен в тот же поток, что и вывод используемого отладчика GDB, то появляется возможность у самой отлаживаемой программы «запутать» процесс отладки. Например, отправив в какой-то момент на свой вывод строку «^exit». Тогда используемая надстройка сделает вывод, что GDB завершил выполнение, и прервет сеанс отладки. Избежать этого можно, как разведя выводы отлаживаемой программы и GDB, так и применяя для каждой команды MI случайно сгенерированные числовые токены, описанные в п. 3.1.

В-третьих, пользователь или разработчик надстройки над GDB должен самостоятельно проверять и устанавливать набор символов (character set), используемых в GDB и/или в отлаживаемой программе, чтобы иметь возможность правильно обрабатывать вывод GDB.

6. Заключение

В работе был рассмотрен машинно-ориентированный текстовый интерфейс отладчика GDB. Несмотря на указанные недостатки, авторы рекомендуют использовать его при разработке собственных или адаптации существующих надстроек для коммуникации с GDB. Кроме того, средства контролируемого выполнения также могут использовать этот интерфейс для полуавтоматического взаимодействия с интерактивным отладчиком, своевременно выявляя ошибки, отказы или атаки, и обеспечивая, тем самым, выполнение целевыми комплексами своих задач.

В дальнейшем планируется адаптация интерфейса для графической надстройки отладчика реального времени, входящего в состав комплекса контролируемого выполнения.

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

Machine Oriented Text Interface to GDB

Vladimir Galatenko, Konstantin Kostiukhin

Abstract. The GDB interactive debugger is the main debugging tool for programs developed on the Linux platform. While providing ample opportunities for debugging, GDB, however, is not able to meet all the needs of, for example, multiprocessor systems developers. One of the solutions is the GDB/MI text interface discussed in this article, with which third-party developers can extend the functionality of the debugger.

Keywords: debugging, multiprocessor systems, CLI, GUI, GDB, MI

Литература

1. GDB: The GNU Project Debugger, https://www.sourceware.org/gdb/.
2. The GDB/MI Interface, https://sourceware.org/gdb/onlinedocs/gdb/GDB_002fMI.html#GDB_002fMI.
3. DDD: The Data Display Debugger, https://www.gnu.org/software/ddd/.

Особенности сборки кросс-компилятора GCC и бинарных утилит

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

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

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

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, sambor@niisi.ras.ru

Аннотация. Разработчики, использующие свободно распространяемый компилятор GCC, нередко сталкиваются с проблемой правильной сборки кросс-компилятора и необходимых бинарных утилит из исходных текстов для заданной целевой архитектуры. В статье приводится общая последовательность действий по самостоятельной сборке кросс-компилятора и бинарных утилит для архитектуры MIPS, подходящая для разных версий компилятора. Приводятся примеры проблем, которые могут при этом возникать, и способы их решения.

Ключевые слова: компилятор, бинарные утилиты, GCC, сборка, MIPS.

1. Введение

Свободно распространяемый компилятор GCC [1] широко используется разработчиками встраиваемых систем и систем реального времени, функционирующих под управлением широкого спектра аппаратных платформ.

В данной работе будет приведена апробированная авторами инструкция по правильной сборке кросс-компилятора и необходимых бинарных утилит для архитектуры MIPS. При необходимости эта инструкция может быть взята за основу при сборке компилятора для других аппаратных платформ.

2. Предварительные действия

Прежде всего надо убедиться, что доступно необходимое программное обеспечение. Для получения архивов исходных текстов может понадобится утилита wget, а для их распаковки - tar. Если планируется работа с git-репозиторием, то, естественно, нужен сам git.

Точный список необходимых программ и диапазон допустимых версий этих программ зависит от того, какие версии компилятора и бинарных утилит предполагается собирать. Обязательно потребуются следующие утилиты:

make
gcc (обязательно с поддержкой C++)
makeinfo (из пакета texinfo)

Также могут понадобиться утилиты bison, flex, m4.

Если сборка происходит под управлением операционной системы Linux, то bash и стандартный набор команд (cd, rm, mkdir, и т.п.) в системе уже есть. Библиотеки GMP, MPFR и MPC заранее устанавливать не нужно, так как их лучше собирать самим (см. п. 3).

Для удобства сборки компилятора требуется создать каталог, например, gcc-mke-build, и перейти в него (имя gcc-mke-build произвольно и дальше не будет использоваться). Затем в этом каталоге удобно создать подкаталог для исходных текстов, например, src.

```
mkdir gcc-mke-build
cd gcc-mke-build
mkdir src
```

Далее создадим каталог, в который будут установлено все собранное программное обеспечение:

```
sudo mkdir /opt/gcc-mke
```

Следует учесть, что для записи в /opt обычно требуются права администратора, поэтому использована команда sudo. Ниже все команды установки запускаются через sudo.

При необходимости, путь /opt/gcc-mke можно заменить на более удобный, используется он только в ключах команд configure. Например, если невозможно использовать sudo или нежелательно устанавливать компилятор в общедоступное место, то можно установить его в домашнем каталоге, но при этом рекомендуется использовать полный путь (начиная с ‘/’, а не с ‘~’). Для непродолжительных экспериментов можно установить компилятор в каталог временных файлов, например, в /var/tmp/gcc-mke.

3. Сборка арифметических библиотек

Для сборки компилятора (но не бинарных

утилит) требуются три библиотеки: GMP – функции для операций над целыми числами неограниченной длины, MPFR – функции для операций с произвольной точностью над числами с плавающей запятой и MPC – аналогичные функции для операций над комплексными числами с плавающей запятой. Библиотека MPFR использует функции библиотеки GMP, а библиотека MPC, в свою очередь, использует функции из MPFR, поэтому собирать их нужно именно в таком порядке.

3.1. Сборка GMP

Сперва необходимо скачать и распаковать актуальную версию библиотеки с официального сайта:

```
wget https://gmplib.org/download/gmp/gmp-6.2.1.tar.xz
cd src
tar xaf ../gmp-6.2.1.tar.xz
```

В команде распаковки (tar xaf ...) параметр ‘a’ означает автоматическое определение компрессора, которым сжат tar-архив. Это удобно, но может не сработать, если в системе не установлен достаточно современный tar, или нет соответствующего компрессора. В этом случае следует дополнительно установить необходимые пакеты или произвести распаковку на другом компьютере.

Далее надо сконфигурировать библиотеку:

```
cd gmp-6.2.1
./configure --disable-shared --enable-static \
--disable-assembly --host=`./configfsf.guess` \
--prefix=/opt/gcc-mke
```

Ключ --prefix=/opt/gcc-mke указывает конфигуратору каталог, куда будет установлена библиотека, ниже он будет использован для всех команд configure. Смысл этого и остальных ключей можно узнать, запустив ‘./configure –help’ или, при необходимости, посмотрев непосредственно сам файл configure. Использование ключей --disable-assembly и --host в данной команде обсуждается ниже в разделе 6.

После конфигурирования производим компиляцию и сборку библиотеки:

```
make -j5
```

Ключ -j используется для параллельной сборки, значение ключа следует устанавливать в зависимости от количества ядер процессора и размера оперативной памяти.

Если памяти достаточно, то рекомендуется указывать число N+1, где N - число ядер, обычно его можно узнать в /proc/cpuinfo. Если памяти совсем мало, то использование параллельной сборки не рекомендуется. Также использовать -j не рекомендуется для команд make check и make install.

Проверить успешность сборки можно следующей командой (все тесты должны проходить успешно):

```
make CFLAGS="-O0" check
```

Здесь используется отменяющий оптимизацию параметр CFLAGS="-O0". Связано это с тем, что есть отдельные версии GMP и отдельные версии GCC, такие что с оптимизацией один из тестов компилируется неправильно и дает ложную ошибку (это не означает наличие ошибки в GMP).

Установка собранной библиотеки GMP (файл libgmp.a) с заголовочными файлами и документацией производится командой:

```
sudo make install
cd ../..
```

3.2. Сборка MPFR

Надо скачать актуальную версию библиотеки с официального сайта и распаковывать ее:

```
wget https://www.mpfr.org/mpfr-current/mpfr-4.1.0.tar.xz
cd src
tar xaf ../mpfr-4.1.0.tar.xz
```

При конфигурировании необходимо указать путь к установленной ранее библиотеке GMP:

```
cd mpfr-4.1.0
./configure --disable-shared --enable-static \
--prefix=/opt/gcc-mke \
--with-gmp=/opt/gcc-mke
```

Оставшиеся действия аналогичны п. 3.1:

```
make -j5
make check
sudo make install
cd ../..
```

3.3. Сборка MPC

Получение и распаковка актуальной версии библиотеки:

```
wget --no-check-certificate \
https://www.multiprecision.org/dowloads/mpc-1.2.0.tar.gz
cd src
tar xaf ../mpc-1.2.0.tar.gz
```

Ключ --no-check-certificate потребовался, поскольку на данный момент на сайте

www.multiprecision.org просроченный сертификат.

При конфигурировании следует так же, как и в п. 3.2, указать расположение библиотек GMP и MPFR:

```
cd mpc-1.2.0
./configure --disable-shared --enable-static \
--prefix=/opt/gcc-mke \
--with-gmp=/opt/gcc-mke \
--with-mpfr=/opt/gcc-mke
```

Действия по сборке и установке библиотеки такие же, как и в предыдущих пунктах:

```
make -j5
make check
sudo make install
cd ../..
```

4. Сборка бинарных утилит

Бинарные утилиты и компилятор также можно собирать из архивов с исходными текстами определенной версии, как и библиотеки выше. Этот способ будет приведен позже. А сейчас продемонстрируем сборку из исходных текстов, полученных из git-репозитория.

Сперва следует клонировать репозиторий с бинарными утилитами и gdb (gdb - отладчик, его собирать не надо):

```
cd src
git clone \
git://sourceware.org/git/binutils-gdb.git
```

Далее надо извлечь из репозитория в рабочий каталог нужную нам версию:

```
cd binutils-gdb
git checkout binutils-2_37
cd ../..
```

Затем нужно создать отдельный каталог для сборки бинарных утилит (исходные тексты лежат отдельно) и перейти в него:

```
mkdir binutils-build
cd binutils-build
```

При конфигурировании бинарных утилит необходимо обратить внимание на маршрут к файлу configure и используемый ключ --srcdir):

```
../src/binutils-gdb/configure \
--srcdir=../src/binutils-gdb \
--prefix=/opt/gcc-mke --target=mips-mke-elf \
--disable-gold --disable-gdb --disable-sim
```

Ключи --disable-gold, --disable-gdb, --disable-sim отменяют сборку программ, которые не понадобятся.

После конфигурирования производится непосредственно сборка и установка:

```
make -j5
sudo make install
cd ..
```

Проверка (make check) для бинарных утилит не запускается, так как тестов очень много, и не вся функциональность бинарных утилит имеет смысл для целевой машины без операционной системы. Поэтому не все тесты (по крайней мере для архитектуры MIPS) будут выполнены успешно. Соответственно, интерпретация результатов проверки требует содержательного анализа. То же самое, еще в большей степени, касается проверки компилятора.

Как было упомянуто выше, можно получить исходные тексты бинарных утилит (той же версии 2.37) скачав их с официального сайта, не клонируя весь репозиторий:

```
wget http://ftp.gnu.org/gnu/binutils/binutils-2.37.tar.bz2
```

или

```
wget --no-check-certificate \
https://ftp.gnu.org/gnu/binutils/binutils-2.37.tar.bz2
```

(опять просроченный сертификат).

Затем распаковываем архив в каталог src:

```
cd src
tar xaf ../binutils-2.37.tar.bz2
cd ..
mkdir binutils-build
cd binutils-build
```

Надо обратить внимание, что распакованный каталог с исходным текстами имеет другое имя: не binutils-gdb, а binutils-2.37. Поэтому потребуется немного модифицировать команду конфигурации, запускаемую в binutils-build:

```
../src/binutils-2.37/configure \
--srcdir=../src/binutils-2.37 \
--prefix=/opt/gcc-mke --target=mips-mke-elf \
--disable-gold --disable-gdb --disable-sim
```

(или переименовать binutils-2.37 в binutils-gdb в каталоге src).

Далее производится сборка и установка бинарных утилит:

```
make -j5
sudo make install
cd ..
```

5. Сборка компилятора

Также, как и для бинарных утилит, следует начать с клонирования репозитория (вариант с распаковкой архива определенной версии будет приведен ниже):

```
cd src
git clone git://gcc.gnu.org/git/gcc.git
```

Далее извлекается в рабочий каталог требуемая версия:

```
cd gcc
git checkout releases/gcc-11.3.0
cd ../..
```

Затем надо создать каталог для сборки компилятора (исходные тексты лежат отдельно) и перейти в него:

```
mkdir gcc-build
cd gcc-build
```

Конфигурируем компилятор:

```
../src/gcc/configure --srcdir=../src/gcc \
--prefix=/opt/gcc-mke --target=mips-mke-elf \
--disable-libssp --disable-gcov --disable-lto \
--enable-languages=c --enable-multilib \
--with-gmp=/opt/gcc-mke \
--with-mpfr=/opt/gcc-mke \
--with-mpc=/opt/gcc-mke
```

Если кроме компилятора C надо собрать также компилятор C++, то следует заменить ключ конфигурации

```
--enable-languages=c
```

на

```
--enable-languages=c,c++
```

и добавить ключ

```
--disable-libstdcxx
```

поскольку невозможно собрать библиотеку libstdc++ (библиотека времени выполнения GNU C++) для целевого процессора без операционной системы.

Затем следует сборка и установка компилятора:

```
make -j5
sudo make install
cd ..
```

Как и в случае с бинарными утилитами можно получить исходные тексты определенной версии компилятора gcc, скачав их с официального сайта без клонирования всего репозитория:

```
wget ftp://ftp.gnu.org/gnu/gcc/gcc-11.3.0/gcc-11.3.0.tar.gz
```

Заметим, что в последней команде использован доступ по протоколу FTP, а не HTTP или HTTPS, как ранее. Какой протокол использовать, зависит от настройки сети и политик безопасности системы, в которой происходит сборка. Кроме того, тот или иной протокол может просто временно (или постоянно) не поддерживаться сайтом, с которого скачиваются исходные тексты.

Затем распаковываем архив в каталог src:

```
cd src
tar xaf ../gcc-11.3.0.tar.gz
cd ..
mkdir gcc-build
cd gcc-build
```

Конфигурируем компилятор, не забыв сменить имя gcc на gcc-11.3.0:

```
../src/gcc-11.3.0/configure \
--srcdir=../src/gcc-11.3.0 \
--prefix=/opt/gcc-mke --target=mips-mke-elf \
--disable-libssp --disable-gcov --disable-lto \
--enable-languages=c --enable-multilib \
--with-gmp=/opt/gcc-mke \
--with-mpfr=/opt/gcc-mke \
--with-mpc=/opt/gcc-mke
```

Дальше так же, как ранее:

```
make -j5
sudo make install
cd ..
```

6. Рекомендации и проблемы

Следует помнить, что собранный компилятор (в отличии от бинарных утилит) нельзя просто копировать из каталога в каталог, так как он запускает свои проходы (cc1, as, и т.д.) по абсолютным маршрутам. Поэтому, если все же по какой-

то причине (например, при оптимизации дискового пространства) требуется переместить компилятор, то необходимо на старом месте поместить символьную ссылку на новый каталог компилятора, чтобы все старые маршруты остались корректными.

Если планируется использовать собранный компилятор разными пользователями на разных компьютерах, то рекомендуется выполнять все процедуры сборки от специального пользователя, созданного для этой цели. Иначе возможны проблемы, например, с тем что у реального пользователя, исполняющего вышеописанные процедуры могут быть нестандартные значения переменных окружения (PATH, LD_LIBRARY_PATH и т.п.).

Также, если планируется использовать компилятор и бинарные утилиты на разных компьютерах следует быть осторожным с разделяемыми библиотеками. В частности, можно заметить, что библиотеки GMP, MPFR и MPC собирались выше только в неразделяемом варианте (с расширением ‘.a’), для того, чтобы они включались в компилятор на стадии компоновки. Тем самым исключается путаница с версиями этих библиотек, которые могут быть уже установлены на компьютере.

Последняя рекомендация не является бесспорной: возможны ситуации, когда именно неразделяемая библиотека ограничивает переносимость собранной с ней программы. При сборке библиотеки GMP в п. 3.1 были использованы ключи конфигуратора --disable-assembly и --host=`./configfsf.guess`. Они нужны для того, чтобы помешать конфигуратору оптимизировать GMP с использованием возможностей конкретного процессора компьютера, на котором происходит сборка (следует заметить, что конфигуратор по умолчанию полагается на информацию от файла сценария config.guess).

```
./configure --disable-shared --enable-static \
--disable-assembly --host=`./configfsf.guess` \
--prefix=/opt/gcc-mke
```

Если не использовать ключ --host возможна ситуация, когда, например, конфигуратор опознает, что сборка библиотеки GMP происходит на компьютере с процессором Skylake и прописывает в файлах сборки (Makefile) использование инструкций из набора BMI2 (Bit Manipulation Instruction set 2). После того как полученная библиотека GMP была использована для сборки компилятора, при дальнейшей попытке запустить этот компилятор на компьютере с процессором Sandy Bridge произошло аварийное завершение компилятора, поскольку Sandy Bridge не поддерживает инструкции из BMI2.

Проблема с оптимизацией под конкретную модель процессора возникает только для библиотеки GMP, при конфигурировании остальных библиотек, а также бинарных утилит и самого компилятора, нет необходимости использовать ключ --host.

Кроме приведенных выше общих рекомендаций, для конкретных версий компилятора и бинарных утилит встречаются отдельные проблемы при их сборке. Например, если вместо использованной выше версии компилятора gcc-11.3.0 собирать более старую, но популярную версию gcc-7.5.0 (последнюю в седьмой ветке), то можно столкнуться со следующей ошибкой. При запуске команды ‘make -j5’ на 64-битном компьютере процесс сборки прерывается при попытке сконфигурировать каталог zlib (содержащий библиотеку сжатия данных). При этом конфигуратор библиотеки zlib выводит следующее сообщение:

```
error: Link tests are not allowed after GCC_NO_EXECUTABLES.
```

Это означает, что конфигуратор прекращает работу, потому что не знает, как собирать исполняемые файлы. Эта ошибка специфична именно для библиотеки zlib.

Причина в том, что файлы сборки содержат указание на компиляцию библиотеки zlib в двух вариантах: 64-битном и 32-битном, при том что 32-битный вариант, как правило, бесполезен, поскольку производится сборка 64-битного компилятора. Проще всего обойти эту ошибку, установив на 64-битную систему, где производится сборка, пакеты, требуемые для сборки 32-битных программ.

Например, в случае системы Linux Fedora Core достаточно при помощи команды dnf установить 32-битные варианты библиотеки glibc.i686 и glibc-devel.i686 (при этом будут установлены еще несколько 32-битных пакетов, необходимых для glibc):

```
sudo dnf install glibc.i686 glibc-devel.i686
```

Осталось заметить, что возможность сборки 32-битных программ на 64-битном компьютере легко проверить, выполнив команду:

```
gcc -m32 hello.c -o hello
```

где hello.c - любая программа на C, например, самая короткая:

```
main(){}
```

Некоторые версии бинарных утилит также

содержат разные ошибки в сценариях сборки. Например, для использованных выше бинарных утилит версии 2.37 в каталоге /opt/gcc-mke/share/man/man1 устанавливаются пустые man-страницы.

Это ошибка именно версии 2.37, исправленная в версии 2.38. Ее можно исправить, применив к исходным текстам бинарных утилит версии 2.37 заплатку (patch), доступную по ссылке [2]. Сделать это можно вручную (просто добавить две строки в texi2pod.pl) или при помощи команды

```
patch -p1 < файл_с_заплаткой
```

Как можно видеть на этих примерах, при сборке различных версий бинарных утилит и компилятора могут возникать разные проблемы, требующие внимательного изучения. В следствии этого, полная автоматизация процесса сборки возможна только для конкретных версий, после «ручной» сборки и проверки корректности и переносимости полученных инструментов.

7. Заключение

В каталог /opt/gcc-mke установлены бинарные утилиты и компилятор для целевой архитектуры mips-mke-elf. Исполняемые файлы компилятора и бинарных утилит (ассемблер, компоновщик, objdump и др.) находятся в каталоге /opt/gcc-mke/bin.

Кроме исполняемых файлов, включаемых файлов и библиотек в /opt/gcc-mke установлена документация в формате info, для ее просмотра необходимо правильно задать переменную окружения INFOPATH:

```
INFOPATH=/opt/gcc-mke/share/info info
```

Можно указать в качестве аргумента конкретную программу, например, компилятор:

```
INFOPATH=/opt/gcc-mke/share/info info gcc
```

или компоновщик:

```
INFOPATH=/opt/gcc-mke/share/info info ld
```

Также в /opt/gcc-mke/share/man устанавливаются man-страницы, для их просмотра необходимо указывать имя команды, под которым она помещена в каталог bin, например, mips-mke-elf-gcc для компилятора:

```
MANPATH=/opt/gcc-mke/share/man man mips-mke-elf-gcc
```

Описанный выше подход позволяет собирать разные версии бинарных утилит и компилятора. Для этого можно просто выполнить команду ‘git checkout' для другой версии. В статье описана сборка актуальных на 2021 год версий бинарных утилит и компилятора, но если планируется собирать компилятор и бинарные утилиты более старых версий, то могут потребоваться другие версии библиотек GMP, MPFR, MPC. Рекомендуемый согласованный набор старых версий: gmp-4.3.2, mpfr-2.4.2, mpc-0.8.1.

Разумеется, нельзя в одном тексте описать все, что касается сборки компилятора и бинарных утилит. Например, сознательно опущены вопросы, связанные с подписями скачанных архивов и их проверкой. Принципиально описывалась только «ручная» процедура, без попыток автоматизировать этот процесс сценариями на языке shell, общим файлом сборки make или с использованием утилиты rpmbuild.

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

Building the GCC Compiler and Binary Utilities as Cross Tools

Vladimir Galatenko, Galina Levchenkova, Sergej Samborskij

Abstract. Developers using the freely distributed GCC compiler often face the problem of correctly building the compiler and the necessary binary utilities from source texts, as well as their configuration to the desired target architecture. The article provides a general sequence of actions for building of a cross-compiler and binary utilities for the MIPS architecture, suitable for different versions of the compiler. Examples of problems that may arise in this case and ways to solve them are given.

Keywords: compiler, binary utilities, GCC, building, MIPS

Литература

1. GCC: The GNU Compiler Collection, https:// https://gcc.gnu.org/.
2. Binutils commit, https://sourceware.org/git/?p=binutils-gdb.git;a=commit;h=2dad02b6

Логические ошибки в подсистемах ввода-вывода современных операционных систем

А.Б. Бетелин[1], А.А. Прилипко[2], Г.А. Прилипко[3], С.Г. Романюк[4], Д.В. Самборский[5]

[1]ab@niisi.msk.ru, [2]aaprilipko@niisi.msk.ru, [3]prilipko@niisi.msk.ru, [4]sgrom@niisi.ras.ru, [5]samborsky_d@fastmail.com

[1,2,3,4,5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия

Аннотация. Современные операционные системы используют уровни абстракции для описания интерфейса ввода-вывода данных. Построенные по этим принципам операционные системы обычно соответствуют стандарту POSIX, описывающему синтаксис и семантику системных вызовов. Тем не менее, применение стандартов не гарантирует согласованности программных компонент и корректности использованных алгоритмов. В данной работе описано несколько логических ошибок, типичных для сложных систем, разные части которых разрабатывались независимо. Анализ описанных ошибок дает основания утверждать, что детальное описание программных моделей систем ввода-вывода для их формальной верификации и тестирования создаваемого кода могут значительно увеличить надежность операционных систем и прикладных программ.

Ключевые слова: файловая система, RAID-массив, логическая ошибка, формальная верификация, тестирование

1. Введение

Стандарт интерфейса POSIX операционных Unix-подобных систем описывает системный интерфейс операционной системы, в том числе операции с файлами и каталогами [1]. С момента своего создания более 30 лет назад стандарт POSIX в некоторой степени эволюционировал, но в целом изменения были небольшими. В стандарте POSIX так и не были описаны промежуточные уровни системы ввода-вывода, необходимые для построения различных систем хранения данных. Внутреннее устройство ядра операционной системы не было предметом стандартизации, поскольку задачей стандарта POSIX было создание общего интерфейса для ОС Unix разных производителей. Кроме того, на момент написания первых версий стандарта POSIX операционные системы семейства Unix были устроены значительно проще, чем современные ОС.

Например, типичная конфигурация файлового сервера ОС Linux включает в себя уровни файлового кэша (page cache), файловой системы (интерфейс virtual file system, VFS), системы управления логическими томами (logical volume management, LVM), программного RAID-массива и подсистем драйверов SCSI или NVMe. Соответственно, каждая операция записи данных проходит через стек этих уровней и может быть завершена как подтвержденной записью в устройства накопителей, так и записью в файловый кэш или в очередь отложенных операций драйверов блочного ввода-вывода. Прикладная программа имеет только ограниченные средства контроля за порядком выполнения этих операций, основные параметры подлежат настройке на системном уровне. Поэтому для настройки оптимальной производительности подсистемы ввода-вывода или диагностики возникающих в ней системных ошибок могут потребоваться глубокие знания об архитектуре ОС и специальные диагностические инструменты [2]. Описания алгоритмов работы разных уровней подсистемы ввода-вывода и драйверов устройств часто не перекрываются между собой и сводятся к техническому описанию деталей реализации, что усложняет понимание того, как работает ОС в целом в случае возникновения нештатных ситуаций.

Если бы проектирование ОС выполнялось с использованием формальных спецификаций, то каждый уровень программного интерфейса мог бы формулировать необходимые условия для входных параметров и, со своей стороны, гарантировать выполнение определенных обязательств. Когда эти условия и обязательства детально описаны, появляются возможности верификации — как с помощью формального доказательства недостижимости некорректных состояний, если алгоритмы тестируемых компонентов достаточно просты, так и с помощью функциональных тестов, имитирующих разнообразные штатные и аварийные ситуации.

Ниже приведены примеры логических ошибок и ситуаций неопределенного поведения в программных комплексах, которые были обнаружены на стадии эксплуатации, хотя могли быть обнаружены ранее, на стадиях проектирования и разработки.

2. Потеря данных в дисковых массивах RAID5/6

Чтобы обеспечить отказоустойчивость, массив RAID5 объединяет блоки данных в группы и дополняет каждую группу дополнительным блоком четности, в котором хранится результат побитовой операции XOR, суммирования по модулю 2 битов данных этих блоков. Блоки одной группы расположены на разных дисках, это дает теоретическую возможность полного восстановления данных при отказе одного из этих дисков. Аварийная ситуация ошибки записи, называемая также "write hole", возникает в массиве RAID5 в случае одновременного выхода из строя одного из дисковых устройств и аварийной перезагрузки системы во время записи данных. Предположим, что в случае минимального массива RAID5, состоящего из трех дисков, в блоках одной группы располагаются данные A1, B1 и A1+B1, соответственно, где A1+B1 обозначает данные блока четности с побитовой суммой данных блоков A1 и B1. Если в момент аварийного отключения были обновлены данные блока B, а блок четности обновиться не успел, то состояние данных блоков станет A1, B2 и A1+B1. При одновременном выходе из строя первого диска окажется, что данные A1 полностью потеряны. Действительно, попытка их восстановления будет выполняться по неверной формуле A1+B1+B2 вместо верной формулы A1+B2+B2=A1, которая бы применялась, если блок четности успел бы обновиться. Ситуация усугубляется, когда блок A1 содержит служебные данные файловой системы или другие важные данные, не связанные с адресатом записи. Потеря этих данных может быть обнаружена позднее, когда они понадобятся прикладным программам или операционной системе, поэтому ошибки такого вида называют скрытыми повреждениями данных или "silent data corruption". Стратегия журнализирования записей файловой системы в этом случае помочь не может.

Вероятность одновременного системного сбоя и выхода из строя одного из дисков на практике оказывается выше ожидаемой, поскольку эти события могут иметь общие причинно-следственные связи. Ошибки этого вида могут произойти и при большем количестве дисков, а также в случае использования RAID6.

Аналогичный сбой в массиве RAID1, т.е. в режиме простого дублирования данных, приведет только к потере последней записи. При этом выполняемая в момент сбоя операция записи не подтверждена, поэтому любая прикладная программа, ответственно подходящая к хранению данных (т.е. использующая синхронизацию записи вызовом fsync или выполняющая запись в синхронном режиме O_DSYNC), должна будет вызвать повторную запись. Целостность файловой системы при таком сбое обеспечит журнал, в котором будет храниться информация для восстановления последнего согласованного состояния структур данных.

Для предотвращения ошибок вида "write hole" в 2015 году ядро Linux было модифицировано: в исходном коде MD-драйвера, реализующего функцию дисковых массивов RAID5 и RAID6, была добавлена поддержка журнала записи [3] на отдельном устройстве. Таким образом, данная проблема была решена только в версии Linux 4.4 – через 15 лет после появления функции поддержки дисковых массивов RAID5 и RAID6. Позже, в версии Linux 4.12, был также добавлен механизм "partial parity log" (PPL), сохраняющий старые версии блоков данных и блоков четности в служебных структурах разметки дисков RAID-массива. Включение механизма PPL снижает производительность, в худшем случае на 30-40% [4], но зато предотвращает аварии вида "write hole" без необходимости создания журнала записи.

Пример этой ошибки демонстрирует, что если обновление данных становится хотя бы немного более сложным, чем простое копирование, как в RAID1, то попытки избежать журнализирования неизбежно приводят к потерям данных. На эту опасность теоретики систем управления базами данных указывали еще более 40 лет назад, см. "UPDATE IN PLACE: A poison apple?" [5].

3. Останов работы RAID1-массива в случае сбоя работы одного устройства

Учитывая вышеописанные опасности использования дисковых массивов RAID5/6, можно предположить, что уровни RAID1 и RAID10 обеспечат надежную и бесперебойную работу. Действительно, алгоритм работы на этих уровнях гораздо проще, поэтому они обеспечивают и сохранность данных, и повышенную скорость чтения. Тем не менее, программные RAID-массивы ядра Linux могут не оправдывать ожидания бесперебойной работы во всех случаях неисправности дисковых устройств. Алгоритм ра-

боты MD-драйвера подразумевает, что устройство либо выполнит операцию чтения или записи, либо сообщит об ошибке, но не остановит свою работу на неопределенное время. Т.е., если диск не ответил на запрос, то весь RAID-массив будет находиться в состоянии ожидания, что для пользователей хранилища данных будет выглядеть как отказ, который может привести к ошибкам в прикладных программах после истечения таймаутов, и, в конечном счете, к потере данных. Данная ситуация не была предусмотрена при разработке кода MD-драйвера, хотя она, очевидно, демонстрирует нестыковку уровней системного кода.

Было предложено два способа решения этой проблемы. Во-первых, можно обеспечить включение аппаратных таймаутов в устройствах накопителей, вызывающих завершение операции с кодом ошибки, чтобы программный уровень RAID-массива мог пометить это устройство как отказавшее и продолжить работу в неполном составе. Позже администратор может диагностировать это устройство, и либо заменить, либо повторно ввести в эксплуатацию. Для устройств с интерфейсом SATA/SAS, предназначенных для использования в серверах, такая настройка должна присутствовать (так называемые TLER, ERC, или CCTL таймауты). Во всех популярных дистрибутивах ОС Linux настройка этих таймаутов не выполняется автоматически при включении дисков в RAID-массивы, хотя соответствующая поправка к пакету smartmontools была предложена 6 лет назад [8]. Во-вторых, несколько позже в MD-драйвере для уровней RAID1 и RAID10 была предложена поддержка атрибута таймаута "failfast" [6, 7].

Эти меры могут исключить полный останов программного RAID-массива, однако просто необычно долгие задержки в работе устройств, не превышающие заданные таймауты, все же могут существенно снизить среднюю скорость работы ввода-вывода. В случае использования магнитных дисковых накопителей рекомендуется периодическая сверка копий данных RAID-массива, которая вызывает тестирование поверхности дисков и перенос сбойных секторов данных, которые иначе могли бы вызвать снижение скорости работы устройства. Для SSD-накопителей, показывающих нестабильное время выполнения операций, обычно нет других решений кроме обновления внутренней прошивки или замены более надежными и, как правило, более дорогими моделями накопителей.

Если дисковый массив используется для хранения данных виртуальных ОС, то среда виртуализации также могла бы использовать таймаут и, например, запланировать приостановку виртуальной системы в случае недоступности устройства. Тем не менее, многие популярные среды виртуализации не предоставляют такой возможности. Например, у QEMU/KVM есть атрибут error_policy для драйверов блочного хранилища, но он предусматривает только реакцию на ошибки ввода-вывода и ситуацию исчерпания свободного места в хранилище виртуальных дисков.

4. Неверная интерпретация семантики функций синхронизации сервером СУБД PostgreSQL

В марте 2018 года разработчики СУБД PostgreSQL обнаружили, что сбой записи данных на диск в ОС Linux может остаться незамеченным и привести к потере данных сервером баз данных [9]. Причиной этих ошибок было то, что семантика системных вызовов fsync и fdatasync не соответствовала ожиданиям разработчиков PostgreSQL. Эти системные вызовы нужны для принудительной синхронизации записи данных указанного файлового дескриптора, т.е. для сброса страниц файлового кэша и буферов ввода-вывода на постоянный носитель информации. Они используются, если файл открыт в обычном режиме (без опций O_DSYNC и O_DIRECT), и программе необходимо получить гарантию, что данные записаны на носитель и не будут потеряны при сбое или аварийном отключении сервера.

В случае неудачной попытки выполнить запись данных на носитель функция fsync возвращает код ошибки EIO. Но в документации ОС Linux не сообщается, что происходит с незаписанными блоками данных. Оказалось, что ядро ОС Linux в этом случае очищает данные, как будто они были записаны, и поэтому повторный вызов fsync для этого файлового дескриптора уже не сообщает об ошибке. Программа PostgreSQL в случае неудачи повторяла вызов fsync и, убедившись в его успешном завершении, считала данные записанными, а первую ошибку считала временным сбоем устройства. В ответ на запрос о причинах такого поведения разработчики ядра Linux пояснили, что данная функция сообщает только об ошибочных ситуациях, возникших после последнего, а не последнего успешного вызова fsync. Из этого следует, что повторный вызов fsync не имеет смысла в ОС Linux даже для тех накопителей данных, которые могут быть временно недоступны, а потом восстановиться после сбоя.

Такой алгоритм работы ядра ОС Linux был выбран потому, что чаще всего подобные ошибки вызывались неожиданным отключением USB флэш-накопителей, когда трудно ожи-

дать, что устройство будет возвращено и продолжит работу, а накопление незаписанных данных привело бы к переполнению оперативной памяти и сбою всей системы. Однако известно, что некоторые POSIX-совместимые системы, например, ОС FreeBSD, все же не делают такую очистку данных. Поэтому в ОС FreeBSD повторные вызовы fsync могут инициировать успешную запись данных, если устройство накопителя данных вернулось к нормальной работе, или продолжают возвращать код ошибки EIO, если устройство все еще не готово к приему данных.

Если переданные для записи данные были потеряны, то прикладной программе в среде ОС Linux остается либо повторить все операции записи с момента последнего успешного вызова fsync (т.е. вызовы функций write, writev, aio_write, и др.), либо аварийно завершить работу, тем самым передав проблему на более высокий уровень. Разработчики PostgreSQL выбрали второй вариант, поскольку при рестарте сервера СУБД будет прочитан журнал запланированных записей (write ahead log, WAL) и восстановлено последнее корректное состояние базы данных [10]. Если же ошибка EIO возникала во время записи в сам журнал WAL, то в этой ситуации и более ранние версии сервера PostgreSQL сразу сигнализировали об ошибке и завершали свою работу. Кроме того, операции записи в журнал WAL в среде ОС Linux по умолчанию выполняются в синхронном режиме, т.е. используется опция O_DSYNC вызова open, и поэтому вызовы fsync по отношению к ним не нужны.

Данный пример демонстрирует, что даже в предельно простой ситуации, когда в среде одной ОС работает одна прикладная программа и выполняет буферизованную запись на одно устройство, то все равно не удается обойтись без семантических неопределенностей на стыках уровней абстракций.

5. Алгоритм использования внутреннего кэша записи SSD-накопителей

Все модели SSD-накопителей серверного применения имеют область кэш-памяти, которая позволяет откладывать запись данных во flash-память, тем самым достигая высокой производительности даже для операций записи не выровненных и сильно фрагментированных данных. Алгоритм использования этой кэш-памяти также тесно связан с уровнем отображения логических блоков данных в страницы flash-памяти, поэтому такое кэширование необходимо не только для увеличения производительности, но и для экономии ограниченного ресурса перезаписи страниц памяти устройства накопителя.

При этом производители большинства моделей SSD-накопителей гарантируют сохранность всех отложенных для записи данных при внезапной потери энергопитания. Эта функция называется Power Loss Protection (PLP) и обеспечивается встроенной конденсаторной батареей, питающей накопитель в течении времени, необходимого для записи данных кэша в энергонезависимую flash-память.

Такая функция фактически стирает различие между кэш-памятью и flash-памятью. Поэтому некоторые модели SSD-накопителей игнорируют команды синхронизации кэша записи, которые драйвер накопителя посылает в ответ на системные вызовы fsync, инициированные прикладной программой. Другие же модели, наоборот, в ответ на эти команды скрупулезно выполняют запись всех отложенных для записи данных. По-сути, оба варианта поведения имеют смысл, поскольку есть свобода интерпретации смысла системного вызова fsync. Второй вариант предпочтителен, когда нужно гарантировать, что данные не будут потеряны, даже если накопитель затем подвергнется временному сбою по какой-либо причине — это может быть сбой микропрограммы, импульс в цепи электропитания или другое внешнее воздействие ("космические лучи"). Но обычно пользователь накопителя подразумевает, что ему достаточно гарантии сохранности данных при неожиданной потере электропитания. В этом случае для сохранения высокой производительности приложений с частыми вызовами fsync приходится либо использовать только накопители первого типа, либо пробовать программно отключать во внутренних настройках накопителя режим кэширования записи. Тогда накопитель, скорее всего, продолжит выполнять запись данных сначала в кэш-память, поскольку так работает его внутреннее ПО, но драйвер накопителя в ядре операционной системы перестанет посылать накопителю команды сброса данных, потому как будет уверен, что кэширование записи выключено. В результате вызовы fsync перестанут замедлять работу накопителя, т.к. будут игнорироваться.

Если накопитель не имеет функции PLP, то выключение кэширования сильно снизит скорость записи, поскольку тогда не только вызовы fsync, но и обычные операции записи будут ожидать подтверждения записи в микросхемы энергонезависимой памяти. Такие устройства не рекомендуется использовать без дополнительного HBA-адаптера или RAID-контроллера, т.к. при внезапной потере питания в них могут повредиться любые данные, даже не затронутые последними операциями записи. Используемый

HBA-адаптер или RAID-контроллер должен иметь собственную конденсаторную батарею с емкостью, достаточной для выполнения отложенных записей.

Таблица 1. Классификация SSD-накопителей по их заявленным характеристикам и измеренной производительности операций записи

Тип	Функция PLP	Скорость записи при вкл. кэше	Скорость записи при выкл. кэше	Комментарий
1	+	высокая	высокая	Устройство считает, что функция PLP гарантирует надежную запись
2	+	высокая	низкая	Устройство считает, что выключение кэширования требует записи данных во flash-память даже при наличии функции PLP
3	–	высокая	высокая	Устройство игнорирует выключение кэширования, несмотря на отсутствие функции PLP
4	–	высокая	низкая	Устройство не имеет функции PLP и корректно реагирует на выключение кэширования

Оказывается, что для достижения как надежности хранения, так и высокой скорости записи данных помогает следующее эмпирическое правило: нужно выключить кэширование записи в настройках устройства накопителя, измерить производительность операций записи, и если скорость снизилась, то либо использовать другие модели накопителей, либо подключить существующие через адаптер или контроллер, имеющий защиту от потери питания. В последнем случае настройку всех устройств следует выполнять согласно инструкции адаптера или контроллера. Таблица 1 классифицирует накопительные устройства согласно наблюдаемой производительности при включенном и выключенном кэшировании записи. Вышеупомянутое эмпирическое правило не работает только по отношению к устройствам третьего типа, которые скрывают отсутствие функции PLP, и уже поэтому должны быть исключены из рассмотрения. Таким образом, устройства первого типа допустимо использовать непосредственно, а устройства второго и четвертого типов рекомендуется подключать к HBA-адаптеру или RAID-контроллеру.

Для устройств второго типа такое подключение может считаться избыточным, но оно позволяет делегировать функцию кэширования контроллеру, что обычно повышает надежность и производительность. Дополнительно рекомендуется проверить скорость записи больших блоков данных при выключенном кэшировании записи в SSD-накопителе. Если скорость записи в этом режиме экстремально низкая, то это означает, что функция кэш-памяти необходима для нормальной работы устройства. Такие устройства не следует использовать даже при установке в RAID-контроллер, поскольку контроллер может настаивать на выключении кэша записи устройства накопителя.

В тех случаях, когда пользователям необходима надежность записи данных не только при потере электропитания, но и в иных ситуациях (отказ внутреннего ПО, единичный сбой микросхемы контроллера устройства), следует использовать устройства второго типа и выключить кэш записи. Это снизит скорость записи данных, но даст гарантии целостности записанных данных непосредственно после подтверждения операции. В этом случае гарантируется целостность завершенных транзакций записи, но не гарантируется выполнение всех транзакций. Если требуется дальнейшее увеличение степени надежности, то этого можно достичь дублированием устройств записи и обеспечением восстановления верной копии данных программными или аппаратными средствами (например, RAID-массивом).

6. Заключение

Из четырех вышеописанных ситуаций только

первая — потеря данных в массивах RAID5 — ограничена одним программным модулем. В остальных примерах причиной ошибочного или неожиданного поведения ОС и прикладных программ служит или недостаточно точно описанная программная модель подсистемы ввода-вывода, или ее неверная интерпретация, как в случае ошибки СУБД PostgreSQL. Это демонстрирует то, что в любых сложных программных системах могут скрываться ошибки, даже если эти системы имеют большую пользовательскую базу и успешно эксплуатируются в течение десятилетий. Решить эти проблемы могла бы формальная верификация корректности всего программного кода операционной системы, но для больших систем это практически невозможно. Так, пока единственным примером успешной верификации кода ОС является микроядро seL4 [11], имеющее всего 10 тысяч строк исходного кода.

Кроме ошибок системного ПО еще одним источником аварийных ситуаций служит непредсказуемость поведения оборудования. В этих условиях программистам прикладного и системного ПО следует подвергать сомнению спецификации как программных, так и аппаратных интерфейсов, и, при возможности, проверять их соблюдение специальными тестами. Архитекторам программных систем можно рекомендовать строить программные и аппаратные системы из проверенных комбинаций компонентов, т.е. не полагаться на теоретически совместимые, но не проверенные на практике конфигурации.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН (Проведение фундаментальных научных исследований (47 ГП) по теме «1021060909180-7-1.2.1 Развитие методов математического моделирования распределенных систем и соответствующих методов вычисления. (FNEF-2022-0007)»).

Logical errors in the input/output subsystems of modern operating systems

A.B. Betelin, A.A. Prilipko, G.A. Prilipko, S.G. Romanyuk, D.V. Samborskiy

Abstract. Modern operating systems use abstraction layers to describe data input/output interfaces. Operating systems based on these principles conform to the POSIX standard, which describes syntax and semantics of system calls. However, use of standards alone does not guarantee the consistency of used software components and correctness of designed algorithms. This article describes several logical errors which are typical for complex systems, different parts of which were developed independently of each other. An analysis of the described errors allows us to state that definition of the detailed models for software components and their subsequent formal verification and application for code testing can significantly improve robustness of the created operating systems and applications.

Keywords: file system, RAID array, logical error, formal verification, testing

Литература

1. Стандарт POSIX.1-2017. The Open Group Base Specifications Issue 7, 2018 edition IEEE Std 1003.1-2017.

2. Gregg, Brendan. Linux Systems Performance. Portland, OR: USENIX Association, 2019.

3. Сайт "A journal for MD/RAID5", https://lwn.net/Articles/665299

4. Сайт "Partial Parity Log for MD RAID 5", https://lwn.net/Articles/715280

5. J. Gray, et al. The transaction concept: Virtues and limitations, VLDB. – 1981. – T. 81. – C. 144-154.

6. Сайт "Add 'failfast' support for raid1/raid10", https://lwn.net/Articles/706865

7. Сайт "Timeout Mismatch", https://raid.wiki.kernel.org/index.php/Timeout_Mismatch

8. Сайт "Many (long) HDD default timeouts cause data loss or corruption (silent controller resets)", https://www.smartmontools.org/ticket/658

9. Сайт "PostgreSQL's handling of fsync() errors is unsafe and risks data loss at least on XFS", https://lwn.net/Articles/752093

10. Сайт «Руководство PostgreSQL. Часть III. Администрирование сервера. Глава 29. Надёжность и журнал предзаписи», https://postgrespro.ru/docs/postgresql/13/wal-reliability

11. Klein, Gerwin, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski, and Gernot Heiser. Comprehensive formal verification of an OS microkernel. ACM Transactions on Computer Systems (TOCS) 32, no. 1 (2014): 1-70.

Использование при разработке СнК метода программной инжекции сбоев для оценки перехода на технологию с меньшими проектными нормами

П.О. Черняков[1], А.П. Скоробогатов[2]

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

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

Аннотация. В данной работе представлен подход к оценке достаточности применяемых мер парирования сбоев в цифровых блоках систем на кристалле (СнК) при переходе на технологию с меньшими проектными нормами и целесообразности данного перехода при условии сохранения прежнего уровня сбоеустойчивости. Подход основан на методе программной инжекции сбоев и продемонстрирован на примере переноса контроллера внешней статической памяти с технологии с 250 нм проектными нормами на технологию с 65 нм проектными нормами.

Ключевые слова: резервирование, TMP, SEU, SET, инжекция сбоев

1. Введение

Повышение требований к производительности микроэлектронной ЭКБ – один из основных стимулов развития технологий интегральных схем. Перенос проекта микросхемы на технологию с меньшими проектными нормами позволяет достичь лучших характеристик по рабочей частоте и энергопотреблению, а также усложнить функциональность за счет использования большего числа транзисторов на единицу площади. Одной из значимых характеристик схемы, ухудшающихся при переходе на технологию с меньшими проектными нормами, является сбоеустойчивость. Для разработок, предназначенных для использования в космической электронике, при переходе на меньшие технологические нормы необходимо использовать дополнительные меры по парированию сбоев [1]. Тройное модульное резервирование (TMP) является эффективным и одним из наиболее распространенных методов парирования сбоев, однако негативно сказывается на характеристиках схемы, таких как быстродействие и энергопотребление, что может нивелировать преимущества от перехода на меньшие проектные нормы. Кроме этого, разработка и производство микросхемы, а также её испытание на стойкость к одиночным эффектам, требуют больших временных и материальных затрат. Следовательно, необходимо иметь возможность оценки достаточности мер по парированию сбоев и ожидаемых преимуществ от перехода на меньшие проектные нормы ещё на этапе разработки микросхемы.

В данной работе с использованием инструмента для инжекции сбоев на этапе логического моделирования [2], [3] сравниваются реализация контроллера внешней статической памяти, разработанная по технологии 250 нм КНИ, с несколькими реализациями этого контроллера, разработанными по 65 нм объемной технологии, отличающимися различными мерами парирования сбоев. Оценивается выигрыш в быстродействии при переходе на технологию с проектными нормами 65 нм при условии сохранения сбоеустойчивости как минимум на уровне реализации на 250 нм. Данная статья – полная версия работы, представленной на конференции «Стойкость-2021» [4].

2. Тестируемое устройство

В качестве объекта для исследования был выбран СФ-блок контроллера внешней статической памяти, являющийся типовым блоком систем на кристалле и обладающий относительно невысокой сложностью и небольшим размером, что делает его удобным для моделирования. В составе данного блока присутствуют только триггеры и ячейки комбинационной логики и отсутствуют вложенные СФ-блоки и памяти.

Были подготовлены несколько различных топологических реализаций данного блока. Одна реализация по 250 нм КНИ КМОП технологии без дополнительных мер парирования сбоев, а также четыре реализации по 65 нм объемной КМОП технологии:

1) без мер парирования сбоев;
2) с применением локального TMP (ЛТМР): троируются только триггеры,

комбинационная логика, включая ячейки дерева синхросигналов, остается в единственном экземпляре;

3) с применением распределенного TMP (PTMP): троируются триггеры и комбинационная логика, исключая ячейки дерева синхросигналов, остающиеся в единственном экземпляре;
4) с применением глобального TMP (ГTMP): троируются все логические компоненты блока [5].

При разработке реализаций ставилась цель достичь схожей плотности размещения элементов на топологии. Также задавались схожие временные ограничения (constraints) на входные и выходные сигналы в процентах от тактового сигнала. Характеристики получившихся реализаций приведены в таблице 1.

Таблица 1. Основные характеристики тестируемых реализаций

Реализация	**Площадь, 10^5 мкм^2**	**Плотность размещения, %**	**Частота, МГц**
КНИ 250 нм	7,93	77,4	44
Без TMP 65 нм	1,37	74,8	166
ЛTMP 65 нм	2,16	76,7	149
PTMP 65 нм	4,22	78,1	128
ГTMP 65 нм	4,34	77,8	123

3. Тестовая система

В составе тестовой системы, представленной на рисунке 1, можно выделить несколько основных блоков:

1) тестируемое устройство с необходимым тестовым окружением;
2) эталонное устройство с аналогичным тестовому устройству окружением;
3) блок инжекции сбоев;
4) монитор;
5) блок управления.

Тестируемое устройство (ТУ) – одна из реализаций контроллера памяти в виде нетлиста. К тестируемому устройству подключается блок системной шины (БСШ) и модуль памяти, с которыми ТУ взаимодействует в рамках теста.

Эталонное устройство – RTL-модель контроллера памяти. Эталонное устройство соединено с собственными экземплярами памяти и БСШ.

Блок инжекции сбоев или инжектор – специальный программный инструмент, позволяющий при моделировании вносить сбои в элементы нетлиста, в том числе с учетом их расположения. Для внесения сбоев строится квадратная область заданного размера, после чего отбираются ячейки, полностью или частично попадающие в данную область. Половину стороны данной области назовем радиусом внесения сбоев. Таким образом можно моделировать множественные сбои от попадания одной частицы. Сбой в триггере (SEU) моделируется как изменение его логического состояния на противоположное. Сбой в комбинационном элементе (SET) моделируется как временное изменение значения его выхода на противоположное.

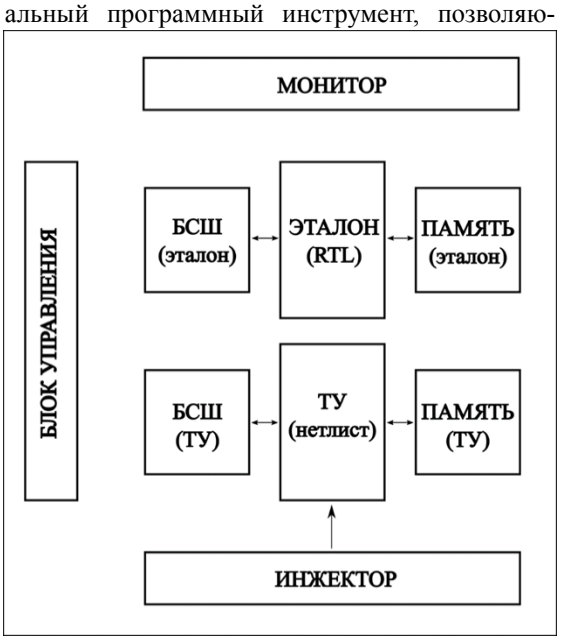

Рис. 1. Структура тестовой системы

Монитор отвечает за сравнение результатов теста у тестируемого и эталонного устройств.

Блок управления отвечает за подготовку и запуск тестов на тестируемом и эталонном устройствах.

В качестве тестов выступают простейшие запросы от блока системной шины к контроллеру памяти на чтение или запись данных.

В каждом цикле работы тестовой системы сначала тест запускается на эталонной модели контроллера, после завершения теста фиксируется состояние блока системной шины и модуля памяти эталонного контроллера, а также продолжительность выполнения теста. Затем этот тест

запускается на тестируемом устройстве. В случайный момент времени с помощью инжектора вносятся сбои в тестируемое устройство, координата для внесения сбоев выбирается случайным образом. По завершению теста состояние блока системной шины и модуля памяти тестируемого устройства сравниваются с состоянием аналогичных блоков эталонного устройства. В случае отличий, а также при зависании тестируемого устройства фиксируется ошибка. После выполнения цикла тестовые окружения тестируемого и эталонного устройств приводятся в одинаковое состояние, а сами устройства сбрасываются. После этого может запускаться следующий цикл. Для каждой реализации контроллера памяти запускались 10^5 циклов тестирования.

4. Выбор параметров для инжекции

Одной из основных сложностей при использовании инжектора сбоев, особенно в случае сравнения реализаций на различных технологиях, является выбор корректных параметров для моделирования: радиуса области внесения сбоев и длительности импульса переходного процесса. Это связано с недостаточностью экспериментальных данных по определению этих характеристик для микросхем, изготовленных по различным технологиям.

Длительность импульса переходного процесса в общем случае зависит от емкостей и сопротивлений элементов в технологии, а также от напряжения питания. Для различных технологий характерная длительность импульса будет отличаться.

Радиус внесения сбоев для 250 нм КНИ технологии обусловлен только радиусом трека частицы. В 65 нм объемной КМОП технологии присутствует также диффузия заряда между транзисторами (эффект «растекания» заряда по подложке), поэтому радиус внесения сбоев будет больше.

В рамках типичной ячейки логического элемента в каждый момент времени не вся площадь является чувствительной областью. Для реализации по объемной 65 нм технологии этим можно пренебречь, поскольку, учитывая типичный размер ячейки, область сбоев одновременно перекрывает большое число ячеек; для реализации по 250 нм КНИ КМОП технологии необходимо вводить поправочный коэффициент.

Для моделирования реализаций контроллера по технологии 65 нм был выбран радиус внесения сбоев 1 мкм и длительность импульса 100 пс [6], сбои вносятся во все отобранные по критерию пересечения с областью сбоев ячейки. В случае технологии КНИ 250 нм рассматривались 2 случая: условно пессимистичный и условно оптимистичный. В условно пессимистичном случае длительность импульса была аналогична технологии 65 нм (100 пс), радиус выбран 100 нм. В условно оптимистичном случае сбои вносились только в триггеры, радиус был выбран равным 1 нм, что равносильно внесению сбоя только при непосредственном попадании в элемент. Дополнительно для оптимистичного случая вводился поправочный коэффициент, который учитывает, что не вся площадь триггера является чувствительной. Коэффициент равен сумме площадей стоков и затворов транзисторов, деленных на 4 и на площадь триггера. Для наиболее распространенного в рассматриваемой топологии триггера коэффициент равен 0,073. Результаты моделирования представлены в таблице 2 и на рисунке 2.

5. Результаты моделирования и анализ

Таблица 2. Значение наибольшего сечения событий

Реализация	**Наибольшее сечение событий, 10^{-6} см^2**	**Частота худшего сечения событий, МГц**
КНИ 250 нм (пессим.)	47,82	20
КНИ 250 нм (оптим.)	3,38	20
Без ТМР 65 нм	15,11	40
ЛТМР 65 нм	9,36	80
РТМР 65 нм	3,59	40
ГТМР 65 нм	1,09	120

В таблице 2 для каждой реализации приведены: наибольшее полученное сечение событий и частота, при котором это сечение получено. На рисунке 2 показаны зависимости сечений событий от частоты для каждой реализации.

По результатам видно, что, по пессимистичной оценке, реализация по КНИ 250 нм оказывается по сбоеустойчивости хуже остальных вариантов.

Если рассматривать оптимистичный вариант оценки реализации по КНИ 250 нм, то он оказывается заметно лучше, чем варианты по 65 нм

без ТМР и с локальным ТМР, а также немного лучше, чем с распределенным ТМР. Но даже оптимистичный вариант КНИ250 заметно уступает варианту по 65 нм с глобальным ТМР. При этом максимальная частота ГТМР почти в три раза выше. Из этого можно сделать вывод, что с точки зрения сочетания производительности и сбоеустойчивости, переход с 250 нм КНИ КМОП технологии на 65 нм объемную КМОП технологию для данного контроллера оправдан, так как при условии использования глобального ТМР можно добиться повышения максимальной частоты примерно в три раза при большей сбоеустойчивости.

По рисунку 2 видно, что отсутствует явная зависимость сечений событий от частоты. Это можно объяснить высокой функциональной сложностью блока, а также тем, что в вариантах по 65 нм большое значение имеют множественные сбои.

Важно отметить, что полученные значения сечений событий имеют сильную зависимость не только от выбора параметров инжектора (радиуса внесения сбоев, длительности импульса переходного процесса и поправочного коэффициента), но также от набора тестов и параметров работы контроллера.

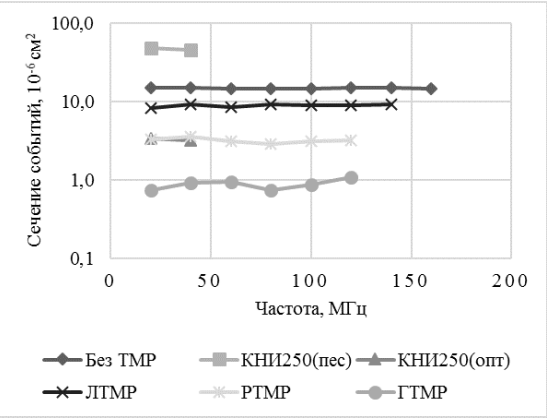

Рис. 2. График зависимости сечения событий от частоты

Рассматриваемый в данной работе СФ-блок контроллера памяти имеет большое число различных настроечных регистров, вариантов подключения памяти и режимов работы с памятью. При этом в тестовой системе использовались лишь самые простые запросы на чтение и запись для одной из возможных конфигураций контроллера. Такие тесты не покрывают всю функциональность и все сочетания различных параметров контроллера. Используя при тестировании другие наборы тестов, можно получить значения сечений, отличающиеся от полученных в данной работе. Поэтому для более точной и надежной оценки предлагается использовать более комплексные наборы тестов, покрывающие не только типичные варианты работы. В том числе можно определить те тесты и режимы, где наблюдаются наихудшие значения сечений событий, и учитывать их в оценке.

6. Заключение

На примере переноса контроллера внешней статической памяти с 250 нм КНИ на 65 нм объемную КМОП технологию представлен подход к оценке целесообразности такого перехода с точки зрения повышения производительности при условии сохранения прежнего уровня сбоеустойчивости.

Для этого были подготовлены и промоделированы с внесением сбоев несколько различных реализаций данного контроллера: реализация по 250 нм КНИ технологии без мер парирования сбоев, уровень сбоеустойчивости которой был принят за референсную точку, и четыре реализации по 65 нм объемной технологии с различными реализациями защиты от сбоев.

По результатам моделирования и анализа был сделан вывод, что такой переход для данного контроллера целесообразен, так как при условии применения глобального ТМР (ГТМР) на 65 нм объемной технологии удалось достичь втрое большей частоты при большем уровне сбоеустойчивости.

Были отмечены основные сложности данного подхода, связанные с обоснованным выбором для различных технологий параметров моделирования: радиуса внесения сбоев, длительности импульса переходного процесса и поправочного коэффициента, учитывающего реальный размер чувствительной области в элементах.

Также было отмечено, что значение сечения событий для одной и той же реализации СФ-блока может отличаться в зависимости от режимов работы и используемых тестов. При выборе оптимальной реализации рекомендуется ориентироваться на более комплексные наборы тестов.

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

Using Fault Injection Technique in SoC Design Flow to Evaluate Shifting to a Smaller Technology Node

P. Chernyakov, A. Skorobogatov

Abstract. This paper presents an approach to assessing the sufficiency of the measures used to mitigation faults in digital blocks of system-on-chip (SoC) during the shifting to technology with smaller technology node and the feasibility of this shifting, provided that the previous level of fault tolerance is maintained. The approach is based on the fault injection technique and is demonstrated by the example of shifting an external static memory controller from 250 nm to 65 nm technology.

Keywords: redundancy, TMR, SEU, SET, fault injection

Литература

1. P. E. Dodd, M. R. Shaneyfelt, J. R. Schwank, J. A. Felix. Current and future challenges in radiation effects on cmos electronics. “IEEE Transactions on Nuclear Science”, V. 57 (2010), No. 4, 1747–1763. ISSN:0018-9499. DOI: 10.1109/TNS.2010.2042613.

2. И.А. Данилов, А.И. Шнайдер (Хазанова), А.О. Балбеков, М.С. Горбунов, А.А. Антонов. Маршрут разработки сбоеустойчивых СБИС с помощью программной инжекции сбоев с учетом топологии. «Вопросы атомной науки и техники. Сер.: Физика радиационного воздействия на радио-электронную аппаратуру», 2019, Вып. 4, 5–10.

3. 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, 289–292. DOI: 10.1109/MIEL.2019.8889643.

4. П.О. Черняков, А.П. Скоробогатов. Баланс сбоеустойчивости и быстродействия при переходе от 250 нм КНИ к 65 нм объемной КМОП технологии на примере контроллера памяти. «Радиационная стойкость электронных систем «Стойкость-2021»: 24-я Всероссийская научно-техническая конференция. Лыткарино, 8–9 июня 2021 г. (тезисы)», Лыткарино, Изд-во Научно-исследовательский институт приборов, 2021, 26–27.

5. M. D. Berg, H. S. Kim, A. M. Phan, C. M. Seidleck, K. A. LaBel, J. A. Pellish, M. J. Campola. The effects of race conditions when implementing single-source redundant clock trees in triple modular redundant synchronous architectures. “Proceedings, 16th European Conference on Radiation and its Effects on Components and Systems (RADECS 2016)”, Germany, Bremen, 2016.

6. A. Evans, M. Glorieux, D. Alexandrescu, C.B. Polo, V. Ferlet-Cavrois. Single event multiple transient (SEMT) measurements in 65 nm bulk technology. “Proceedings, 16th European Conference on Radiation and its Effects on Components and Systems (RADECS 2016)”, Germany, Bremen, 2016.

Графовые нейронные сети и их применение при проектировании цифровых СБИС

Н.В. Желудков[1], К.А. Петров[2]

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

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

Аннотация. В статье рассматривается метод машинного обучения на графах, представлены архитектуры современных графовых нейронных сетей, а также их применение для решения задач проектирования цифровых СБИС, в особенности при размещении стандартных ячеек на этапе топологического проектирования. Решение данной задачи с помощью методов машинного обучения является актуальной проблемой, так как стандартные алгоритмы размещения, используемые в современных САПР, сталкиваются со сложностями при работе с цифровыми схемами, число логических элементов в которых достигает 10^6 и более. Это приводит к длительному времени работы и неоптимальности полученных результатов по параметрам занимаемой площади и энергопотребления проектируемой СБИС.

Ключевые слова: машинное обучение, графовые нейронные сети, GNN, топологическое проектирование СБИС.

1. Введение

В настоящее время часто можно увидеть представление различных данных или информации как в быту, так и в научной сфере в виде графов – математической абстракции, представляющей собой совокупность двух множеств – множества вершин (объектов) и множества ребер (связи и соединения этих объектов). К данным, которые часто представляются в виде графов, относятся представление связи атомов в молекулах различных веществ, связи пользователей внутри социальной сети, множество научных работ с цитированием друг друга и т.д. В проектировании цифровых СБИС в виде графа можно представить нетлист схемы – список логических вентилей и их межсоединений. В таком случае первые будут представлены как узлы графа, а вторые – как его ребра, то есть соединены ли два логических вентиля друг с другом или нет.

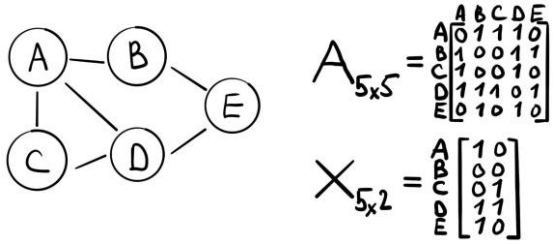

Рис. 1. Матрицы смежности A и признаков X

Очень часто сведения о графе представляют в матричной форме: в виде матрицы смежности A, где a_{ij} элемент равен числу ребер между i и j элементом графа, а также матрицей признаков X, где в i-й строке представлен вектор признаков i-го элемента графа (см. рис. 1).

Для данных, представленных в виде графа, также как для других видов представления информации (видео, изображения, таблицы, текст), можно применить методы машинного обучения для различного типа задач. К таким типам задач можно отнести:

- предсказание на уровне самого графа, то есть задача классификации графа, например, является ли данная молекула токсичной, на основе увиденных ранее экземпляров;
- предсказание на уровне ребер графа - предсказание возможных связей в графе, например, подбор рекламы под пользователя или рекомендации в социальных сетях;
- предсказание на уровне отдельных узлов, например, классификация отдельных узлов, на основе уже обработанных ранее;
- задача кластеризации, то есть задача объединения отдельных узлов графа в группы по набору каких-нибудь признаков.

Простейшая идея обучения на графах заключается в том, чтобы подать на вход полносвязной нейронной сети конкатенацию матрицы смежности A и матрицу признаков X и таким образом обучить модель [1]. Однако такой подход влечет за собой следующие фундаментальные проблемы:

- зависимость от размеров графа и числа его признаков, предложенную выше концепцию нейронной сети можно будет использовать только для графов с одним и тем же числом узлов, к тому же сложность модели будет зависеть от числа узлов, что при большом их числе сильно усложняет модель;
- изменение порядка нумерации узлов влияет на результат обучения; так как для графа нельзя

однозначно определить порядок узлов, то любое изменение в их нумерации влечет за собой изменение элементов матрицы смежности A, что не дает корректно провести обучение;

- граф обладает неевклидовой структурой [2], что, например, сказывается на отсутствие инвариантности к перестановкам узлов, и не позволяет использовать для него стандартные варианты сверточных нейронных сетей CNN.

Таким образом, для решения задачи требуется найти такое представление графа, в котором отражалась бы информация о его структуре и свойствах вершин. При этом такое представление должно вписываться в существующие методы машинного обучения. Такое представление получило название “представление узлов” (Node embedding) – узлам графа ставится в соответствие вектор размерности d (обычно, его размерность меньше числа признаков для узла) в евклидовом пространстве (см рис. 2). Эти представления должны содержать информацию о признаках самого узла и, некоторым образом, содержать в себе информацию о признаках его соседях и об общей структуре графа в окрестности рассматриваемого узла.

Такой аналог свертки для графов позволит применить к представлениям узлов стандартные методы машинного обучения. Естественным образом встает вопрос, каким образом получить данные представления узлов? В последние годы для этой задачи используются новая архитектура нейронных сетей – графовые нейронные сети.

Рис. 2. Представление графа в виде представления его узлов

2. Графовые нейронные сети

Новый подход к получению представлений узлов был предложен в работе [3]. Он заключается в том, что каждой вершине графа ставится в соответствие “скрытое представление” (hidden embedding), которое будет обновляться в зависимости от числа слоев. Нулевое скрытое представление вершины – это ее вектор признаков из матрицы X. Скрытое представление этого узла на первом слое будет представлять собой агрегацию данных от соседей этого узла, которые находятся на расстоянии в 1 вершину, самого скрытого представления рассматриваемого узла на предыдущем слое, а также функции “обновления”, которая вычисляет на основе описанных выше величин обновленное скрытое представление узла в этом слое. Такой подход к агрегации данных от соседей узла и обновлении скрытых представлений получил название нейронная пересылка сообщений (Neural message passing) [4] или сокращенно пересылкой сообщений. Под сообщениями, которыми обмениваются вершины графа, понимают скрытые представления этих вершин на определенном уровне (слое), обработанные определенным образом (выбранным методом или функцией). Проход с номером k (который также можно назвать уровнем или слоем) пересылки сообщений обновляет представление рассматриваемого узла в радиусе k-вершин в его локальном соседстве. В общем виде этот процесс описывается формулой (1):

$$\mathbf{h}_u^{(k+1)} = \mathrm{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathrm{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right) \qquad (1)$$

где $h_u^{(k+1)}$ – представление узла u в слое k+1, $\mathrm{UPDATE}^{(k)}$ – функция, обновляющая представление узла в слое k, $\mathrm{AGGREGATE}^{(k)}$ – функция агрегации данных от соседних узлов, $h_v^{(k)}$ – представление узлов v, которые являются соседями узла u (N(u)) на расстоянии в k узлов.

В зависимости от вида сообщений, способа их агрегации и функции обновления скрытого представления различают несколько видов нейронных сетей. Основной и классической архитектурой сети такого вида является графовая нейронная сеть (GNN, Graph Neural Network). В качестве сообщений здесь используются скрытые представления от соседних узлов в k-радиусе, деленные на степень соответствующего узла (чтобы уменьшить разброс между представлениями вершин, число соседей которых значительно отличается), идущие в финальную формулу с весовыми коэффициентами (которые и являются тренируемыми параметрами). Функция обновления скрытого состояния представляет собой одну из нелинейных функций, используемых в качестве функции активации нейронов в сети – это может быть сигмоидальная функция, функция ReLu и т.д. Формулы (2), (3) и (4) используются для вычисления представлений узлов в GNN:

$$\mathbf{h}_v^{(0)} = \mathbf{x}_v \qquad (2)$$

$$\mathbf{h}_v^{(k+1)} = \sigma\left(\mathbf{W}_k \sum_{u \in \mathrm{N}(v)} \frac{\mathbf{h}_u^{(k)}}{|\mathrm{N}(v)|} + \mathbf{B}_k \mathbf{h}_v^{(k)}\right), \forall k \in \{0..K-1\} \qquad (3)$$

$$\mathbf{z}_v = \mathbf{h}_v^{(K)} \qquad (4)$$

где $h_u^{(0)}$ – начальное представление узла u, равное x_u – u-строчке матрицы признаков, $h_u^{(k+1)}$ – представление узла u в слое k+1, W_k и B_k - веса нейронов в слое k, $|N(v)|$ – степень узла v, z_u – представление узла u в финальном слое K.

Число слоев K является гиперпараметром обучаемой сети и подбирается в зависимости от параметров графа и их особенностей.

После получения финального набора скрытых представлений на k-слое (после прямого распространения), встает задача обучения нашей моделей (параметров W_k и B_k из формулы (3)), чтобы уточнить полученные представления в контексте решаемой задачи. Вид обучения будет зависеть от имеющихся у нас данных (например, формат и тип матрицы признаков X) или типа задачи предсказания. Виды обучения графовой нейронной сети можно разделить на два типа: обучение с учителем и без него.

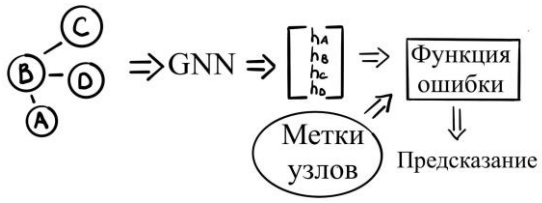

Рис. 3. Обучение с учителем

Обучение с учителем (supervised learning), когда известны, например, виды вершин (каждая вершина имеет метку своего класса, как на рис. 3), на которые надо классифицировать новые. В таких случаях функция потерь считает различие между фактическим значением типа вершины и тем, которое было получено исходя из ее скрытого представления. Такой тип обучения хорошо подходит для задачи предсказания на уровне узлов графа и его ребер.

Обучение без учителя (unsupervised learning) требуется применять в таких задачах, в которых нет меток вершин и ребер. В этом случае обучение можно строить на основе структуры самого графа. Интуитивным является визуализация графа таким образом, чтобы узлы, лежащие близко к друг другу, имели в евклидовом пространстве схожие скрытые представления (см. рис. 4).В данных задачах функция потерь будет оценивать разницу между “близостью” двух вершин на графе и “близостью” представлений этих вершин в пространстве скрытых представлений. В качестве меры близости узлов на графе можно использовать элементы матрицы смежности A или более комплексные оценки, базирующиеся на алгоритмах “случайных блужданий” (DeepWalk и node2vec). В качестве меры близости представлений вершин можно использовать также векторное расстояние между ними или скалярное произведение.

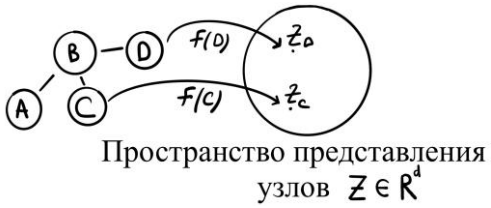

Рис. 4. Представление вершин графа

Отдельно уточним, что на выходе k-слоя графовой нейронной сети мы получаем представления вершин исследуемого графа, прошедших через обученную сеть. Данные представления можно передать на вход другой полносвязной нейронной сети или использовать в качестве входных данных для алгоритма.

Такая архитектура графовой нейронной сети и процесс ее обучения позволяет добиться следующих преимуществ перед примитивным подходом, который рассматривался выше (с подачей на вход сети матрицы смежности и матрицы признаков):

- инвариантность к нумерации узлов;

- позволяет подавать на вход графы любого размера, не требуется заново переучивать сеть.

3. GNN в проектировании цифровых СБИС

Как отмечалось ранее, многие данные при проектировании цифровых СБИС (в том числе нетлист схемы) можно представить в виде графа. Ввиду увеличения функциональной сложности схем и роста числа элементов в них, старые алгоритмы решения многих задач в области проектирования цифровых СБИС показывают неудовлетворительные результаты, а также затрачивают на свою работу много времени и ресурсов вычислительной машины. В последние годы значительно возросло число научных работ, посвященных применению различных методов машинного обучения при проектировании цифровых схем [5]. Особую роль в решении этих задач заняли графовые нейрон-

ные сети. Среди задач проектирования, в которых изучается применение графовых нейронных сетей можно выделить:

- оценка потребляемой мощности на основе GNN (на рис. 5 представлен вариант представления нетлиста в граф, а также нужных параметров узлов и ребер для расчета мощности), представленная в работе [6], где авторам удалось добиться с помощью графовой нейронной сети ускорения работы в 18,7 раза по сравнению со стандартным алгоритмом и ошибкой меньше 5,5%;

- анализ просадки напряжения на сетке земли-питания в работе [7]; оценка просадки питания в цифровом блоке из статьи стандартными средствами одного из коммерческих САПР занимает 3 часа, в то время как на основе GNN – 18 минут с точностью 94%;

- оценка паразитных характеристик при моделировании топологии схемы [8];

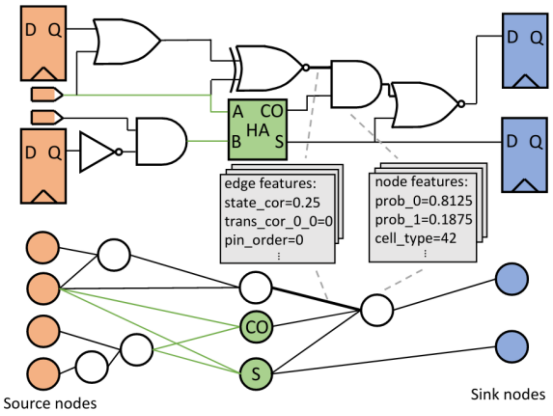

Рис. 5. Представление нетлиста и его параметров в виде графа для оценки мощности [6]

- модель на основе GNN, предсказывающая места на схеме, в которых могут быть проблемы с разводимостью, на основе плана размещения и нетлиста схемы [9]. На рис. 6 на изображении слева представлена реальная оценка разводимости, а справа - предсказанная.

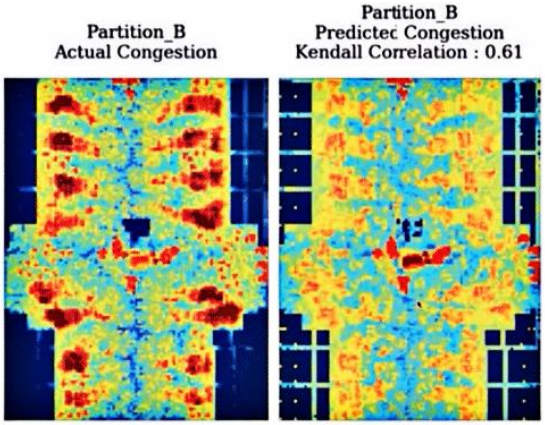

Рис. 6. Оценка разводимости с GNN [9]

К этому списку можно также отнести важную задачу размещения стандартных ячеек. Это задача является одним из наиболее вычислительно сложных этапов топологического проектирования СБИС. После создания плана размещения, включающего в себя определение размеров проектируемого блока, расстановки входных-выходных портов, размещения макроблоков (памятей, аналоговых блоков и т.д.) и построения сетки земли-питания, необходимо разместить стандартные ячейки, представленные в нетлисте. То, как будут размещены ячейки по площади блока, прямым образом отразится на будущее быстродействие схемы, трудности с разводимостью межсоединений на этапе трассировки, а также может затронуть потребляемую мощность и локальные просадки на сетке земли-питания. Этот этап проектирования является критически важным, поскольку от его результата зависят многие важные характеристики проектируемой СБИС.

Большинство алгоритмов, используемых для решения задачи размещения, в своей основе содержат механизм минимизации длины межсоединений ячеек. Годы использования таких алгоритмов показали высокую корреляцию между этой характеристикой и результатами этапа размещения. Например, в САПР с открытым исходным кодом OpenROAD для размещения ячеек используется алгоритм решения аналога электростатических уравнений для размещения методом Нестерова. Недостатком многих подобных методов, как уже упоминалось ранее, является большая длительность работы и качество результатов.

Решение данной проблемы с использованием графовых нейронных сетей предлагается в

работе [10]. Алгоритм, представленный в статье, состоит из двух частей. В первой из оригинального нетлиста схемы необходимо получить граф, узлы которого будут иметь определенные (рассмотренные в дальнейшем) признаки; затем на основе графовой нейронной сети надо получить представление вершин, проведя обучение без учителя этой сети таким образом, чтобы полученные представления вершин имели схожесть с логической близостью элементов в нетлисте. Вторая часть алгоритма заключается в применении к полученным представлениям вершин метода K-средних (алгоритма кластеризации), для объединения “близких” в пространстве представлений вершин в кластеры, которые станут основой для групп-размещения в коммерческом САПР (в работе авторы используют САПР IC Compiler 2 компании Synopsys). Схематически эти два этапа представлены на рис. 7.

Рассмотрим подробнее, как авторы выбирают в своей работе признаки для вершин графа, а также как проводят обучение. Вектор начальных признаков для вершины графа, т.е. изначально для ячейки в нетлисте, формируется на основе иерархической информации – ячейки, находящиеся в одном иерархическом модуле, должны иметь одинаковый вектор признаков, а чем дальше в иерархии друг от друга находятся ячейки, тем сильнее должны отличаться их начальные представления.

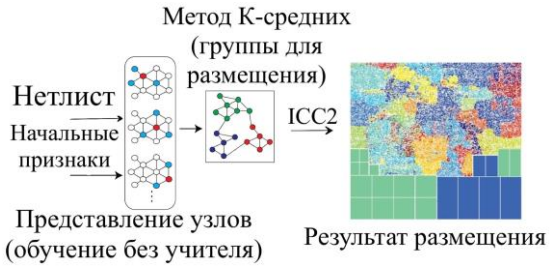

Рис. 7. Схема работы алгоритма [10]

Для кодирования иерархической информации о ячейках используется иерархическое дерево, представленное на рис. 8.

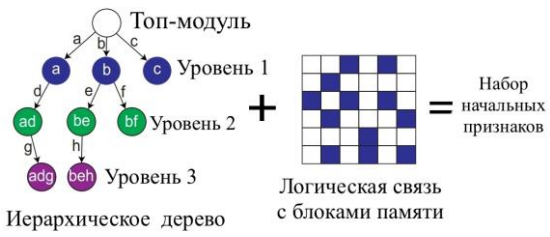

Рис. 8. Задание начальных признаков [10]

Также к общим иерархическим признакам добавляется информация о том, насколько далеко рассматриваемая ячейка находится от блоков памяти (в общем случае – от всех макроблоков в схеме) в нетлисте. Авторы работы аргументируют такой выбор признаков тем, что ячейки, находящиеся в близких иерархических модулях, имеют больше межсоединений между собой, и, если размещать их рядом – это должно позитивно отразиться на качестве этапа размещения. Добавление же к вектору признаков информации о близости к ячейкам памяти обусловлено тем, что пути между триггерами и портами памятей часто являются критическими в плане быстродействия. Добавление данных об этом может помочь провести размещение ячеек с учетом этой особенности.

Для нахождения представлений вершин используется одна из разновидностей архитектур графовых нейронных сетей Graph-SAGE. В отличие от базового варианта GNN, представленного на рис. 4, в котором агрегация сообщений от соседних вершин суммируется с представлением рассматриваемой вершины с некоторыми коэффициентами, в Graph-SAGE агрегация сообщений и представление узла конкатенируются. В своей работе авторы используют 2 слоя такой сети с 128 нейронами в каждом слое.

Для испытания своего метода размещения ячеек авторы использовали два цифровых блока СБИС по технологии TSMC 28nm с числом ячеек около 200 тыс. Новый подход (по сравнению с алгоритмом используемым в САПР ICC2) с использованием графовой нейронной сети позволил сократить суммарную длину проводов на 3,9%, потребляемую мощность на 2,8% и увеличить запас по времени установки (показатель, определяющий общее быстродействие схемы) на 85,7 %.

4. Заключение

В статье рассмотрены особенности представления информации в виде графов (в том числе данных, используемых при проектировании цифровых СБИС) в контексте решения задач машинного обучения, рассмотрены современные архитектуры графовых нейронных сетей, процесс их обучения и достоинства перед другими методами обучения на графах. Проведен обзор современных тенденций в использовании графовых нейронных сетей для решения

различных задач при проектировании цифровых СБИС. Особое внимание уделено работе, затрагивающей использование графовых нейронных сетей на этапе размещения стандартных ячеек во время топологического проектирования СБИС. Использование данного подхода позволило получить лучшие результаты по быстродействию и потребляемой мощности, по сравнению с результатами с использованием стандартного алгоритма в САПР Synopsys ICC2. Основываясь на обзоре описанных выше работ наиболее перспективным направлением исследований в этой области являются работы по решению задачи размещения ячеек:

- исследовать новые способы задания начальных признаков вершин, для более точного соответствия с нетлистом;

- применить разные виды обучения моделей, рассмотреть варианты использования GNN с большим числом слоев для более глубокого анализа;

- найти применение полученным с помощью GNN представлениям узлов для более точной расстановки ячеек, чем при кластеризации.

Исходя из анализа работ на данную тему можно сделать вывод, что данное направление, связанное с использованием графовых нейронных сетей и смежных методов машинного обучения в проектировании СБИС, актуально, его изучение и активное использование позволяет сократить время на разработку интегральных схем, а также получить лучшие характеристики проектируемых СБИС по сравнению со стандартными алгоритмами.

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

Graph neural networks and their application in the design of digital VLSI

N.V. Zheludkov, K.A. Petrov

Abstract. The article discusses the method of machine learning on graphs, presents the architecture of modern graph neural networks, as well as their application for solving digital VLSI design problems, especially in placing standard cells at the layout design stage. The solution of this problem using machine learning methods is an urgent problem, since the standard placement algorithms used in modern CAD systems face difficulties when working with digital circuits, the number of logic elements in which reaches 10^6 and more. This leads to a long operating time and non-optimality of the results obtained in terms of the occupied area and power consumption of the designed VLSI.

Keywords: machine learning, graph neural networks (GNN), layout design of VLSI.

Литература

1. W.L. Hamilton, R. Ying, J. Leskove. Representation Learning on Graphs: Methods and Applications. «Bulletin of the IEEE Computer Society Technical Committee on Data Engineering», Vol. 40 (2017), №3, 52-74.
2. M.M. Bronstein, J. Bruna, Y, LeCun, A.Szlam. Geometric Deep Learning: Going beyond Euclidean data. “IEEE Signal Processing Magazine”, Vol. 34 (2017), №6, 18-42.
3. T.N. Kipf, M. Welling. Semi-supervised Classification with Graph Convolutional Networks. “5th International Conference on Learning Representations 2017”, France, Toulon, ICLR, 2017, 66-80.
4. W.L. Hamilton. Graph Representation Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, Vol. 14 (2019), №3, 1-159.
5. G. Huang, J. Hu, Y. He. Machine Learning for Electronic Design Automation: A Survey. “ACM Transactions on Design Automation of Electronic Systems”, Vol. 26 (2021), №5, 1-46.
6. Y. Zhang, M. Ren, B. Khailany. GRANNITE: Graph neural Network Interference for Transferable Power Estimation. “Design Automation Conference (DAC) 2020”, USA, San Francisco, IEEE, 2020.
7. V. Zhang, M. Ren, B. Keller, B. Khailany. MAVIREC: ML-Aided Vectored IR-Drop Estimation and Classification. “Design, Automation & Test in Europe Conference & Exhibition, 2021”, France, Grenoble, 2021, IEEE, 2021.

8. M. Ren, G. Kokai, W. Turner, T. Ku. ParaGraph: Layout Parasitics and Device Parameter Prediction using Graph Neural Networks, “Design Automation Conference (DAC) 2020”. USA, San Francisco, IEEE, 2020.
9. R. Kirby, S. Godil, R. Roy, B. Catanzaro. CongestionNet: Routing Congestion Prediction Using Deep Graph Neural Networks. “IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC) 2019”, Peru, Cuzco, IEEE, 2019.
10. Y. Lu, S. Pentapati, S.K. Lim. VLSI Placement Optimization using Graph Neural Networks. “IEEE/ACM International Conference On Computer Aided Design (ICCAD)”, USA, San Diego, 2020.

Использование метода косимуляции при разработке высокопроизводительных микропроцессоров

А.Г. Ворсин[1], А.В. Шумаков[2], К.А. Петров[3], П.С. Зубковский[4]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, vorsin@cs.niisi.ras.ru, +7(962)403-51-87;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, shumakov@cs.niisi.ras.ru, +7(916)235-27-67;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, petrovk@cs.niisi.ras.ru, +7(926)145-88-93;

[4]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, zubkovsky@cs.niisi.ras.ru, +7(495)719-78-49

Аннотация. Моделирование, проводимое на низком уровне абстракции при разработке высокопроизводительных микропроцессоров, создает большую нагрузку на вычислительные машины и занимает много времени. Рассмотрен метод косимуляции, позволяющий использовать часть моделируемого микропроцессора в виде абстракции более высокого уровня, тем самым сокращая нагрузку на вычислительные машины и время моделирования.

Ключевые слова: система на кристалле (СнК), высокопроизводительный микропроцессор, RTL-моделирование, программная эмуляция, косимуляция

1. Введение

В процессе разработки системы на кристалле (СнК) высокопроизводительного микропроцессора во избежание накопления ошибок важно как можно раньше и как можно более обширно начать тестировать разрабатываемую модель. Тестирование можно проводить различными способами [1-3] в зависимости от конкретных задач, условий и этапа разработки.

Так, например, функциональное RTL-моделирование позволяет отслеживать и изучать исполнение отдельных логических операций проекта на низком уровне абстракции. С другой стороны, этот способ значительно уступает в скорости менее подробным типам моделирования, так что подходит либо для тестирования не слишком больших устройств, на коротких временных промежутках, либо когда у разработчика нет строгих ограничений по времени [4].

Существуют методы тестирования, при использовании которых части одного моделируемого высокопроизводительного микропроцессора существуют на различных уровнях абстракции либо описаны разными языками. Это может быть ПЛИС-прототипирование в комбинации с RTL-моделированием, Моделирование на уровне транзакций и RTL-моделирование, программная эмуляция аппаратуры и RTL. Все эти методы обозначаются термином косимуляция [5]. Этот подход в определенных случаях имеет превосходство в скорости моделирования и удобстве для разработчика, так как позволяет комбинировать разные уровни абстракции.

Метод косимуляции был применен в процессе тестирования контроллера интерфейса 10-гигабитного Ethernet. В данном случае контроллер был представлен в виде модели, описанной на языке RTL и программно подключен к модели СнК эмулируемой на более высоком уровне абстракции в программе QEMU.

Решение провести тестирование контроллера этим методом было обусловлено необходимостью оперативной проверки на совместимость с драйвером для операционной системы Linux, запуск которой на RTL-модели СнК занял бы неоправданно много времени.

2. Актуальность метода косимуляции

При проектировании СнК и ее составных частей используются различные уровни абстракции. Каждый из них имеет свои преимущества на разных этапах. К уровням абстракции, применяемым при логическом проектировании можно отнести уровень логического нетлиста, уровень RTL и уровень программной эмуляции аппаратуры.

При выборе некомбинированного (проводящегося на одном уровне абстракции) метода тестирования следует учитывать преимущества и недостатки каждого из них.

RTL-модель описывает цифровое устройство в виде последовательных логических операций. Такое описание легко воспринимается человеком и может быть напрямую преобразовано в

модель уровня логических элементов с помощью САПР синтеза.

Компиляция модели – то есть ее сборка САПР на основе исходного кода – занимает порядка нескольких минут и включает в себя составление иерархической структуры и проверку синтаксиса языка.

Напротив, моделирование работы устройства на этом уровне подразумевает определение состояния всех его сигналов в каждом такте и может потребовать значительного времени. Соотношение машинного (времени работы вычислительной системы, на которой производится моделирование СнК) и системного (времени в симуляции) времен может достигать 10^8.

С другой стороны, разработчик получает возможность отслеживать ошибки, сужая зону поиска вплоть до логического элемента, на котором они возникают в любой момент времени.

Уровень логического нетлиста получается путем добавления к RTL описанию информации о задержках сигнала, проходящего через логические элементы и между ними. При моделировании он позволяет производить отладку ошибок, которые не проявляются в RTL, однако его компиляция и сам процесс моделирования на несколько порядков медленнее.

Прототипирование модели СнК микропроцессора на ПЛИС требует предварительной подготовки после каждого изменения структуры исходного кода, так как САПР должна синтезировать файл-прошивку. Благодаря тому, что ПЛИС обрабатывает процессы параллельно, ее производительность сопоставима с производительностью реального устройства. Однако для возможности прототипирования продвинутых устройств со сложной структурой требуются дорогостоящие тестовые стенды.

Программная эмуляция позволяет воспроизводить функции одних вычислительных систем на других. Эмуляция дает возможность взаимодействовать с эмулируемым устройством программными средствами, но при этом отсутствует возможность подробного исследования его внутренней структуры.

Сравнительные характеристики методов тестирования модели микропроцессора при разработке представлены в таблице 1.

Таблица 1. Сравнительные характеристики методов тестирования модели микропроцессора при разработке

Методы / Параметры	RTL моделирование	ПЛИС прототипирование	Программная эмуляция
Время компиляции	Малое	Высокое	Малое
Машинное время моделирования	Высокое	Малое	Малое
Подробность отладочной информации	Высокая	Средняя	Малая
Стоимость аппаратуры для моделирования	Низкая	Высокая	Низкая

Сочетание RTL-моделирования и программной эмуляции в рамках косимуляции позволяет перенять сильные стороны обоих способов в виде высокой скорости тестирования, которая незначительно падает по сравнению с чистой эмуляцией, и высокой подробности внутренних процессов отлаживаемой части. При этом не требуется дополнительных затрат на покупку дорогостоящего оборудования в виде стендов ПЛИС, для проведения косимуляции достаточно персонального компьютера. Стоит отметить, что для реализации данного способа требуется дополнительное время на предварительную подготовку, а также наличие у разработчика знания языков Verilog и C.

3. Используемый эмулятор

QEMU это программное обеспечение с открытым кодом, написанное на языке C, для эмуляции архитектур одних систем на других. Программа поддерживает режимы “пользовательской” и “системной” эмуляции. В первом случае эмулируется только центральный микропроцессор, а во втором и все периферийные устройства [6]. Моделирование в QEMU гораздо быстрее RTL-моделирования, поэтому при необходимости тестирования отдельного устройства в со-

ставе системы перенос основной ее части в эмулятор позволит ускорить ход симуляции в сотни раз. При этом, если оставить тестируемое устройство на уровне RTL, можно будет производить его отладку, не создавая дополнительную нагрузку на вычислительную машину, на которой проводится моделирование.

QEMU поддерживает процессорные архитектуры ARM, MIPS, x86, и другие, а также имеет возможность добавления собственных. При косимуляции контроллера 10-гигабитного Ethernet эмулировалась СнК, разработанная в ФГУ ФНЦ НИИСИ РАН с MIPS-подобной архитектурой процессора. В файле, описывающем этот проект для QEMU, обозначены периферийные модули с указанием базовых адресов, адресных окон и линий прерывания. Устройство, участвующее в косимуляции на уровне RTL, здесь обозначается как “cosim”.

Само устройство “cosim” для QEMU выглядит как описание типов возможных транзакций. Всего 5 типов:

1. Передача запроса тестируемому устройству, с указанием его типа. Запрос может быть чтением или записью в регистр контроллера, или сбросом. В случае записи, помимо адреса целевого регистра контроллеру передадутся данные. Запрос типа сброс приведет к аппаратному сбросу контроллера.
2. Ответ от контроллера. Приходит после каждой передачи запроса и свидетельствует о том, что запрос обработан.
3. Прямой доступ в память (DMA) для записи, где контроллер выступает в роли мастера.
4. DMA чтение из памяти, где контроллер выступает в роли мастера.
5. Передача сигнала прерывания от контроллера.

Описание транзакций выполнено в виде функций, вызываемых согласно алгоритму драйвера или бареметального теста.

4. Обеспечение связи между ПО и RTL-моделью

Связь QEMU с тестируемым устройством осуществлялась посредством интерфейса DPI (direct programming interface) [7]. Этот интерфейс специально предназначен для взаимодействия языков проектирования Verilog и SystemVerilog с языками программирования. Он был реализован в виде вызова функций, подключаемых с помощью библиотеки libdpi.so, которые распознаются как эмулятором QEMU, так и компилятором RTL-кода.

Схема взаимодействия RTL-модели моделируемого контроллера и эмулируемой модели СнК представлена на Рис. 1. Функции библиотеки libdpi.so помещают получаемые при вызове данные в FIFO файлы, которые еще называют “named pipe” каналами. Они предварительно создаются в директории симуляции командой makefifo. На каждое устройство в косимуляции, создается 4 таких канала: для чтения и записи инициализируемых QEMU, и чтения и записи инициализируемых контроллером. Такое количество каналов необходимо для того, чтобы при чередовании различных типов передач данные не перемешивались.

Со стороны QEMU с библиотекой libdpi.so взаимодействует модуль “extcom”. Его копии подключаются отдельно для каждого косимулимуемого устройства и выполняют преобразование формата данных между библиотекой и QEMU, а также помещают и забирают данные из FIFO.

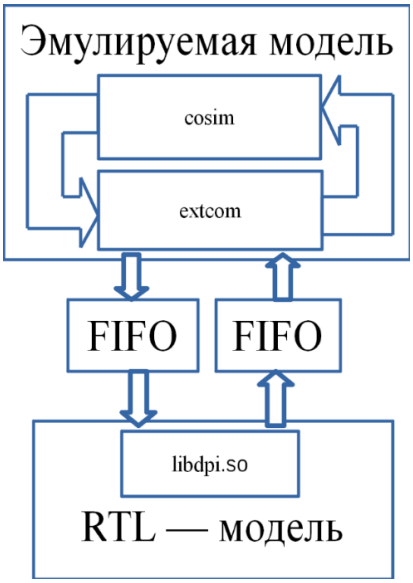

Рис. 1. Схема взаимодействия RTL-модели и эмулируемой модели СнК

5. Подготовка RTL к косимуляции

Для возможности взаимодействия с QEMU, со стороны RTL был создан модуль-обертка в который импортируются функции из libdpi.so. В этот модуль помещается модуль верхнего уровня контроллера, а сигналы его интерфейса взаимодействия с регистрами управляются поведенческими блоками синхронной логики, где выполняется преобразование данных между интерфейсом и функциями библиотеки. Таким образом взаимодействие QEMU и контроллера синхронизируется по тактовой частоте, определяемой на уровне модуля обертки.

6. Результаты

В качестве эксперимента был составлен бареметальный тест, в котором контроллер 10-гигабитного Ethernet работает в режиме петли с прямым доступом в память, результаты которого для двух методов моделирования (RTL-моделирование и косимуляция) представлены в таблице 2.

Таблица 2. Сравнение методов моделирования

	RTL моделирование всей СнК	Косимуляция RTL (контроллер) + QEMU (СнК)
Системное время, нс	367	19
Машинное время моделирования, с	1491	4
Объем задействованной оперативной памяти ЭВМ, МБ	1483	67

Данные из таблицы 2 были получены средствами САПРа. Для данного блока машинное время, необходимое для завершения теста в косимуляции меньше времени при RTL моделировании примерно в 350 раз, при том, что максимальный объем оперативной памяти занятый САПРом для моделирования меньше в 22 раза. Стоит отметить, что память, которая потребовалась для запуска QEMU не учитывалась.

Были успешно завершены тесты, затрагивающие различный функционал контроллера, а также отданы команды на передачу пакетов данных с помощью драйвера для операционной системы Linux.

7. Заключение

В настоящее время возрастающая сложность и размер структур СнК высокопроизводительных микропроцессоров требуют как можно более подробного их тестирования на ранних этапах проектирования. В связи с этим возрастает время тестирования, а также требования к вычислительным машинам, связанные с повышенной нагрузкой на них.

Косимуляция позволяет на ранней стадии разработки СнК начать отлаживать взаимодействие между аппаратным и программным обеспечением. Производительность этого метода позволяет использовать для тестирования сложное ПО (вплоть до запуска операционной системы).

В тоже время, исполнение отлаживаемой части в виде RTL-модели позволяет производить детальную отладку тестируемого устройства, а повторная компиляция при внесении изменений в аппаратный код не занимает много времени.

Было успешно проведено тестирование RTL-модели контроллера интерфейса 10-гигабитного Ethernet в составе СнК, эмулируемой программой QEMU. В процессе тестирования был достигнут 350-кратное снижение машинного времени моделирования, а также 22-кратное снижение потребляемой программой симулятором оперативной памяти.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН по теме FNEF-2022-0004.

Using the Cosimulation Method in High-Performance Microprocessors Development

A.G. Vorsin, A.V. Shumakov, K.A. Petrov, P.S. Zubkovskiy

Abstract. In the development of high-performance microprocessors, low abstraction level modeling creates a large load on computing machines and takes a lot of time. A simulation method that allows to reduce load on computers and simulation time using a part of the simulated microprocessor in the form of a higher-level abstraction is considered in this article.

Keywords: System on Chip (SoC), high-performance microprocessor, RTL-modeling, program emulation, cosimulation.

Литература

1. Robert G. Sargen, Verification and validation of simulation model, IEEE 2011.
2. Harry D. Foster, 2018 FPGA Functional Verification Trends, IEEE 2018.

3. Ney Calazans, Edson Moreno, Fabiano Hessel, Vitor Rosa, Fernando Moraes, Everton Carara, From VHDL Register Transfer Level to SystemC Transaction Level Modeling: a Comparative Case Study, IEEE 2003.

4. Maddu Karunaratne, A. Sagahayroon, RTL fault modeling, IEEE 2005.

5. Claudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, Co-Simulation: A Survey, 2018 ACM Computing Surveys.

6. Fabrice Bellard, QEMU, a Fast and Portable Dynamic Translator, DBPL 2005.

7. IEEE Standard for SystemVerilog — Unified Hardware Design, Specification, and Verification Language, 901-915, IEEE 2013.

Исследование эффективности применения различных архитектур нейросетей в расчете маски в задаче инверсной фотолитографии

Я.М. Карандашев[1,4], Г.С. Теплов[2,3], В.В. Керемет[5], А.А. Карманов[3], А.В. Кузовков[2], М.Ю. Мальсагов[1]

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

[2]АО «НИИМЭ», Зеленоград, Россия, gteplov@niime.ru;

[3]Московский физико-технический институт, Долгопрудный, Россия;

[4]Российский университет дружбы народов, Москва, Россия;

[5]Московский государственный университет им. М.В.Ломоносова, Москва, Россия

Аннотация. В данной работе нами решается обратная задача вычислительной фотолитографии. Расчет топологии маски производился глубокими нейронными сетями. Исследование было направлено на сравнение эффективности нейросетевых архитектур U-Net, Erf-Net и Deep Lab v3, а также встроенных алгоритмов Calibre Workbench в решении задачи инверсной фотолитографии. Обучение искусственных нейронных сетей было выполнено на специально сгенерированном и размеченном наборе данных. Случайные фигуры генерировались с помощью САПР Calibre Workbench для маски затворов транзисторов технологии 90 нм. Сравнение производилось по параметрам точности и скорости. В качестве метрик в работе были использованы edge placement error (EPE) и intersection over union (IOU). Применение нейронных сетей позволило в 100 раз ускорить расчет маски при сохранении точности в 92% на метрике IOU.

Ключевые слова: вычислительная фотолитография, инверсная литография, сверточные нейронные сети, коррекция оптической близости, U-Net, Erf-Net, Deep Lab v3, перевод картинки в картинку, автоматизация электронного проектирования, искусственный интеллект, EPE, IOU

1. Введение

Современные процессы производства микросхем включают технологические операции фотолитографии. Тип операций и используемые методы фотолитографии зависят от минимального линейного размера применяемого в микросхеме и объемов производства. На уровне технологии 180 нм и менее (90 нм, 65 нм, 45 нм, 32 нм) в процессе экспонирования топологии микросхем волновая природа источника излучения с длиной волны 193 нм вызывает эффекты оптической близости [1]. Данный класс эффектов может приводить к искажениям и ошибкам в топологии микросхемы, что в итоге снижает количество выхода годных изделий [2]. К негативным эффектам относятся: укорачивание линий на концах, скругление углов, возникновение перемычек между областями, уширение линий, сужение линий располагающихся в группе и многие другие[3]. Современные фотолитографические технологии включают набор методов повышения разрешающей способности, которые позволяют скорректировать эффекты оптической близости [4]. В число этих методов входят методы преднамеренного изменения маски для компенсирования нежелательных эффектов оптической природы света. Корректировка эффектов оптической близости при топологических нормах 90 нм и 180 нм может осуществляться двумя технологиями. Первая технология - Optical Proximity Correction (OPC), основана на правилах [5] или оптической модели [6]. Вторая технология - Inverse Lithography Technology (ILT), основана на оптимизации параметров окна процесса фотолитографии [7]. Расчет с применением ILT требует значительного времени и вычислительных ресурсов. Проблема усугубляется увеличением вычислительной сложности выполнения расчета для комплекта фотошаблонов одного изделия, что становится особенно критично при снижении минимальной топологической нормы [8]. Смена нормы приводит к увеличению количества фотошаблонов в комплекте изделия и увеличению времени расчета одного слоя [9].

Возможным решением проблемы для OPC и для ILT может служить применение технологий искусственного интеллекта на основе машинного обучения [10] и нейронных сетей[11]. Ряд коллективов ведет исследования по объединению ML-OPC подхода с техническими решениями EDA для ускорения подготовки комплекта

фотошаблонов [12]. Основные направления применения машинного обучения и нейронных сетей в вычислительной фотолитографии рассматривались в работе [13]. Ранее в своей работе мы исследовали эффективность применения различных алгоритмов ML для выполнения OPC [14]. В данной работе фокусом нашего исследования является применение искусственных нейронных сетей для расчета топологии фотошаблона сопоставимого с фотошаблоном рассчитанным ILT.

Одним из вариантов технического решения является применение GAN для расчета фотошаблона. Указанный подход может применяться как отдельно [15], так и в сочетании с последующим классическим решением [16], которое ускоряет обучение или точность вычислений. Другой подход базируется на применении модифицированных архитектур сверточных нейронных сетей [17, 18]. Последний подход использует специально разработанные для решения данной задачи архитектуры [19] или их модификацию [20].

В своей работе мы рассматриваем технологический процесс 90 нм. Имеющиеся в нашем распоряжении OPC-рецепты учитывают в своем расчете параметры фотолитографического оборудования и свойства применяемого резиста, что ограничивает применение сторонних моделей машинного обучения и нейронных сетей. Поэтому нами были сгенерированы оригинальные наборы данных с применением Calibre Workbench. Цель нашего исследования заключалась в определении эффективности существующих архитектур сверточных нейронных сетей по параметрам точности и скорости расчета. В сравнении использовались архитектуры U-Net [21], Erfnet [22] и DeepLabv3+[23].

2. Описание наборов данных, аппаратных ресурсов и программных средств

Исследование было выполнено на двух сгенерированных наборах данных. Генерация наборов данных для обучения сетей была выполнена в CAD Calibre Workbench. Обучающий пример состоит из изображения топологии (рис. 1a) до расчета маски технологией ILT для этой топологии и рассчитанных фотошаблонов (рис. 1b и рис. 1c). В генерации случайных фигур топологии использовались OPC-рецепты для технологии 90 нм. Фотошаблон соответствует наиболее сложному для вычислений этапу – уровню формирования затворов транзисторов. В обоих случаях во время предобработки и разметки данных из изображения целевого вектора удалялись sub resolution assisted figures (SRAF). Изображения приведены к бинарному формату через объединение каналов RGB. Черный цвет соответствует фигуре, а белый цвет является фоном. Разрешение изображений 768x768 пикселя, где каждому пикселю соответствует размер топологии 5x5 нм. Линейные размеры сгенерированных фигур находятся в диапазоне от 0.5 до 4 мкм. Выборка состояла из 1536 пар изображений в формате png. Выборка разделена на следующие 3 части: 70% – данные для обучения, 10% – валидационные данные для определения переобучения, 20% тестовые данные для оценки модели.

Первый набор данных включал исходную топологию рис. 1a и результат ее моделирования без применения опции соответствия MRC(mask rule check), рис. 1b. Первый набор данных применялся на стадии оценки эффективности CNN в качестве алгоритма расчета ILT без опции MRC. Архитектура применяемая в проверке гипотезы – U-Net.

Во втором наборе данных целевой вектор является результатом расчета по технологии ILT с опцией соответствия MRC, рис. 1c. Процесс изготовления фотошаблонов имеет свои технологические ограничения, что приводит к необходимости применения опции соответствию MRC при расчете методом ILT. Второй набор данных использовался для тестирования эффективности архитектур U-Net, ErfNet и DeepLabV3+ по параметрам скорости и точности вычислений.

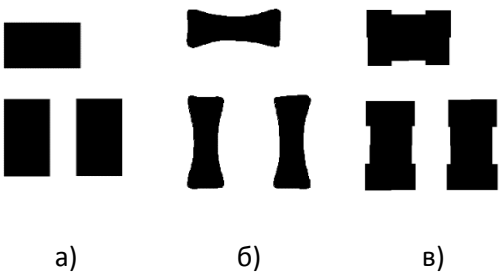

Рис. 1. Рисунки целевых векторов. а) исходный вектор, б) целевой вектор без MRC, в) целевой вектор с MRC.

Расчет фотошаблона в CAD Calibre Workbench с опцией MRC произведен на CPU Intel Gold 6254. Общее время расчета составило 30 минут, что приблизительно соответствует 1 секунде на изображение. Для первичного анализа эффективности применения CNN для решения задачи вычислительной фотолитографии использовался ресурс Google Colaboratory [24] со следующими характеристиками: 12.69 Gb оперативной памяти, 1 видеокарта Nvidia Tesla K80 с 12 Gb видеопамяти. Обучение нейронных сетей U-Net, ErfNet и DeepLabV3+ для решения обратной задачи на датасете с MRC выполнено на GPU Nvidia RTX 3080 laptop version. В обоих

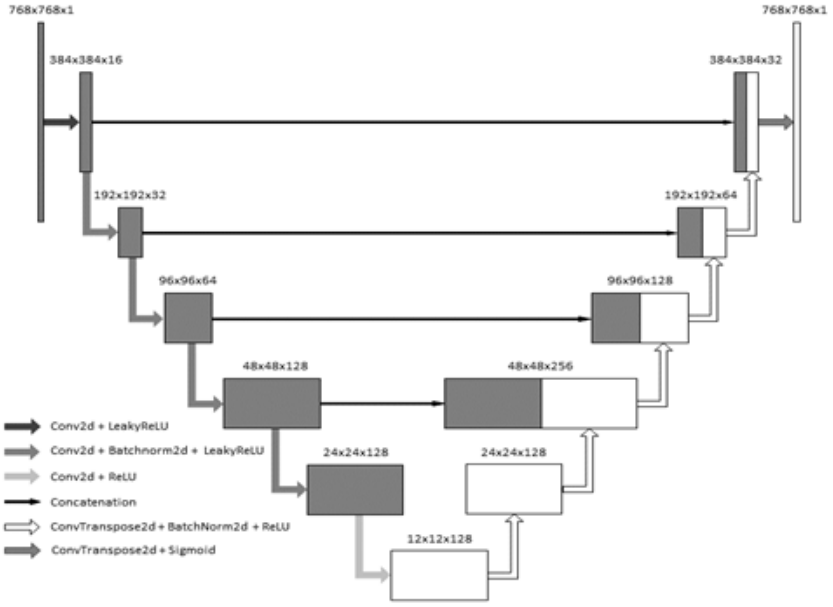

Рис. 2. Архитектура нейронной сети Unet.

случаях использовался фреймворк PyTorch [25].

2.1. Архитектуры нейросетевых моделей

Unet. В качестве первой применяемой для решения задачи архитектуры искусственной нейронной сети была выбрана UNet [26], которая хорошо зарекомендовала себя в задачах сегментации и перевода картинки в картинку [27]. Наша сеть имеет сходную с Unet архитектуру, приведенную на рис. 2. Она состоит из сужающегося пути слева (encoder) и расширяющегося пути справа (decoder).

Сужающийся путь Encoder — типичная архитектура сверточной нейронной сети, – представляет собой повторное применения сверток (4×4), с Striding (2, 2), padding (1, 1) за которыми следуют BatchNorm и функция активации ReLU. В отличие от оригинальной UNET модели, мы отказались от maxpooling слоев, заменив их на понижающих размер Striding. Также, в отличии от оригинальной статьи, мы используем padding, для того чтобы выход нашей сети не отличался размером от выхода, и мы не теряли информацию при обрезке. Также мы используем BatchNorm, которого не было в оригинальной работе.

Decoder состоит из повторяющихся операций повышающих дискретизации карты свойств, после чего объединяется с соответствующим образом карты свойств из стягивающегося пути, после каждой операции следуют BatchNorm и функция активации ReLU, кроме последней, после последней операции следует функция активации sigmoid, чтобы выход сети был в интервале от 0 до 1.

Как уже упоминалось ранее, наш набор данных представляет из себя пару изображений размера 768×768. Мы специально привели картинки к такому размеру, чтобы длина и ширина делилась на степень двойки, в данном случае на 2^8, что необходимо для применения 6 последовательных операций свертки 4×4 с шагом 2. Отсюда следует предложенная архитектура нейросети (рис. 2). Общий размер сети составлял 1 829 025 параметров.

ERFNet. Вторя использованная нами нейронная сеть - это ERFNet. Архитектура сети полностью соответствует предложенной в работе [22]. Основным преимуществом применяемой архитектуры является применение особого блока последовательных сверток – non-bottleneck-1D, который позволяет уменьшить количество параметров в Encoder не сокращая число сверточных слоев. Применяемый в архитектуре Decoder с малым количеством развертывающих слоев также позволяет уменьшить общее количество параметров сети. В качестве основного преимущества сети традиционно указывают соотношение точности и скорости обработки изображений в сравнении с другими применяемыми архитектурами. Общий размер сети составлял 2 063 130 параметров.

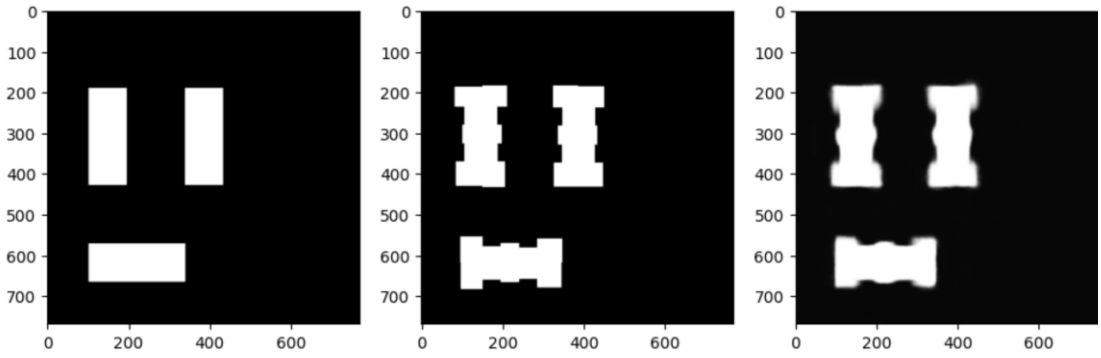

Рис. 3. Характерный результат обучения архитектур. Слева направо: входной вектор, целевой вектор, выход нейронной сети.

DeepLabv3+. Третьей архитектурой для решения задачи была выбрана DeepLabv3+. Архитектура может реализовываться различными вариантами модулей. Сравнительный анализ эффективности модулей проводился ранее в работе [23]. В качестве преимуществ указывается, что разреженные сверточные слои позволяют сильно увеличить перцептивную область нейрона, а сшивка представлений, полученных с различной областью восприятия нейрона,позволяет нейросети обучаться как локальным зависимостям, так и располагающимся в обширной области изображения. Различные варианты архитектуры сети включают следующие основные блоки: backbone, assp, decoder и несколько дополнительных. В исследовании была использована архитектура нейронной сети, которая состояла из трех модулей: модуля backbone, модуля ASPP, и модуля decoder. В нашем исследовании в модуле backbone использовалась предобученная нейронная сеть ResNet 101. Общий размер сети DeepLabv3+ составлял 59 332 642 параметров.

3. Результаты

3.1. Результаты обучения для набора данных ILT без опции MRC

В качестве архитектуры использовалась архитектура Unet, показанная на рис. 2. В качестве функции потерь для обучения использовалась бинарная кросс энтропия, которая вычисляется в каждой точке и определяется:

$$BCE(p,t) = \sum_j -t_j \ln p_j - (1-t_j)\ln(1-p_j) \quad (1)$$

где t - реальное бинарное значение пикселя, p - число от 0 до 1, предсказанное нейросетью.

Для оценки качества получаемых изображений использовалась метрика IoU:

$$IoU(A,B) = \left|A \cap B\right| \Big/ \left|A \cup B\right| \quad (2)$$

Метрика IoU (Intersection over Union, формула (3)), известная также как Jaccard index, — число от 0 до 1, показывающее, насколько у двух объектов совпадает внутренний “объем”. Метрика IoU рассчитывалась между данными с выхода нейронной сети и вектором рис. 1c, в зависимости от используемого набора данных для обучения. В ходе проведения 10 различных запусков процесса обучения были получены следующие усредненные результаты представленные в Таблице 1.

В таблице показано, что все архитектуры сетей продемонстрировали одинаковые показатели качества с точностью до второго знака. Учитывая множественность запусков можно заключить, что точность рассматриваемых моделей совпадает. Стоит отметить, что архитектура UNet обладает самым малым временем расчета, которое вдвое меньше времени расчета ErfNet и в четыре раза меньше времени расчета DeepLabv3+. Таким образом, можно заключить что наибольшей производительностью среди рассмотренных моделей обладает архитектура UNet. Стоит отметить, что расчет одного элемента классическими методами на процессоре Intel Xeon Gold 6254 составил порядка 1000 мс.

К основным недостаткам рассчитанных ответов можно отнести скругление краевых линий в областях углов, заметное невооруженным глазом, рис. 3. На рисунке фигуры обозначены белым цветом, и можно видеть как у выходного изображения нейронной сети наблюдается размытие краев. Данная картина была характерна для всех трех архитектур.

Таблица 1. Усредненные результаты по 10 запускам обучения

Параметры/ Архитектуры	**U-Net**	**ErfNet**	**DeepLab V3+**
IOU	0,921	0,923	0,924
Время расчета одного элемента, мс	7	13	27

Поскольку полученные результаты совпадают с результатами расчетов классическими методами с разницей в чуть более 1%, была проведена оценка качества полученного решения с применением метрики EPE.

EPE — значение ошибки размещения ребра фрагмента в контрольной точке [28] (см. рис. 4 “Смещение”, ошибка выделена черным). Распределение величины EPE (Edge Placement Error) характеризует ошибку размещения ребер фрагментов в смоделированном контуре, которая вычисляется в контрольных точках. На Рисунке 4 можно видеть абстрактную схему выполнения ILT. “Топология” соответствует входному вектору для нашей нейронной сети и аналогична рис. 1а. “Коррекция” соответствует выходному вектору и аналогична рис. 1б. Коррекция представляет собой целевой вектор (рис. 1b и рис. 1c). Для расчета EPE данные с выхода нейронной сети преобразовывались в GDSII, после чего осуществлялось моделирование в САПР “Calibre OPCpro”. Расчет включал в себя 300 смоделированных с помощью нашей нейросетевой модели фигур. Расчет EPE выполнялся только для анализа первого набора данных (без MRC) из-за высокой трудоемкости процесса.

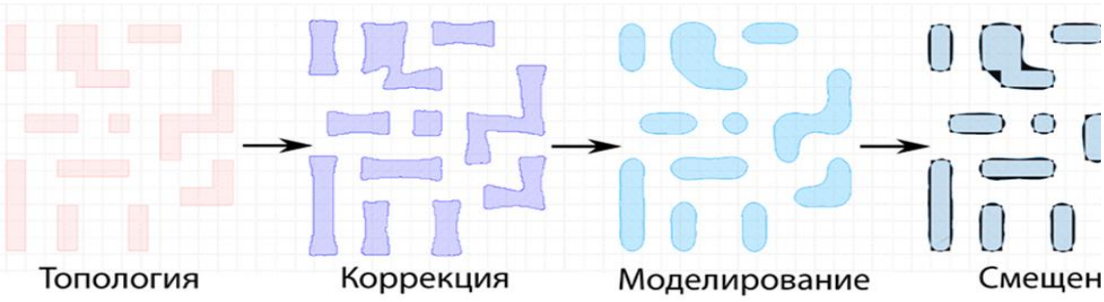

Рис. 4. Процедура получения фигуры смещения смоделированного контура для расчёта распределения величины EPE (Edge Placement Error) [28]. EPE выделена черным на стадии “Смещение”.

3.2. Результаты обучения для набора данных ILT с MRC

Обучение на первом наборе данных производилось с использованием функции потерь попиксельной бинарной кросс энтропии. В качестве метрики применялась IoU и MSE (mean square error). Таблица с процессом обучения представлена в Таблице 2. Модель обучалась за 30 эпох, время прохождения каждой эпохи в среднем 2 минуты. Метрика IoU показала точность в 0.989.

Таблица 2. Обучение нейронной сети по эпохам.

№ Эпохи	**Train loss BCE**	**Val loss BCE**	**IoU**	**MSE**
1	0.22669	0.15972	0.89204	0.00262
10	0.02240	0.02286	0.98000	0.00047
20	0.01661	0.01806	0.98460	0.00037
30	0.01089	0.01290	0.98935	0.00027

Оценка качества нашей модели с помощью метрики EPE, рассчитанной в САПР показана на рис. 5. Качество представлено гистограммой ошибок EPE нейронной сети и EPE оригинальной модели ILT.

Из рис. 5 видно, что распределение ошибок нейронной сети ближе к нулю, т.е. нейросетевая модель дала даже лучший результат, чем оригинальный алгоритм инверсной фотолитографии. Кроме того, время расчета одного изображения с помощью данного пакета САПР занимало секунду, в то время как работа нейронной сети порядка 17 мс, т.е. достигаемое ускорение в вычислении – на два порядка.

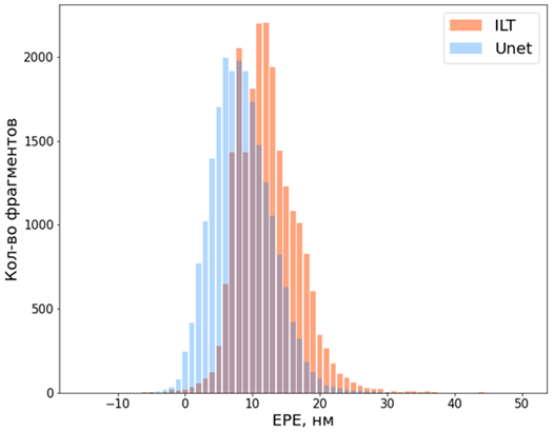

Рис. 5. Гистограммы ошибки EPE Unet и EPE оригинального алгоритма ILT.

4. Заключение

Задачи инверсной фотолитографии, как и другие обратные задачи физики и математики, представляют особый класс задач, в которых применение нейронных сетей кажется наиболее естественным. Действительно, решение, предъявляемое нейронной сетью может быть с легкостью проверено прямым численным моделированием, которое не занимает столь много времени, как итерационный процесс решения обратных задач.

Решение задачи ILT без опции MRC при работе с первым датасетом продемонстрировало результаты превосходящие в скорости и точности применяемые классические алгоритмы, которые являются стандартом в промышленности. Полученная в работе нейросетевая модель UNet на валидационной выборке из 300 пар изображений показала лучший результат, чем оригинальная модель инверсной фотолитографии, при этом время расчета одной структуры уменьшилось на два порядка. Можно заключить, что данная задача решена на приемлемом уровне.

Решение задачи ILT с опцией MRC при работе со вторым датасетом продемонстрировало недостаточную точность работы сетей, которая составила порядка 92% (по метрике IOU). Основным недостатком в результатах рассмотренных моделей является наличие размытия краев у фигур, что особенно заметно вблизи углов фигур. Полученное решение задачи требует дополнительного исследования для устранения этого недостатка.

Поскольку экстенсивный путь увеличения количества слоев модели (применение ErfNet) и общего количества параметров и слоев (применение DeepLabv3+), не возымел эффекта – изменение подхода к обучению сети и применение других сверточных слоев видится нам следующим логичным шагом в достижении результата.

Работа выполнена при поддержке гранта РФФИ No 19-29-03030 «Поиск физических, технологических и схемотехнических нейросетевых решений на основе мемристорных кроссбаров»

Application of Various Neural Networks Architectures for Mask Calculation in Problem of Inverse Photolithography

Ya.M. Karandashev, G.S. Teplov, V.V. Keremet, A.A. Karmanov, A.V. Kuzovkov, M.Yu. Malsagov

Abstract. In this work, we solve the inverse problem of computational photolithography. The mask topology was calculated using deep neural networks. The study was aimed at comparing the effectiveness of the neural network architectures U-Net, Erf-Net and Deep Lab v3, as well as the built-in Calibre Workbench algorithms in solving the problem of inverse photolithography. Training of artificial neural networks was performed on a specially generated and labeled data set. Random shapes were generated using the Calibre Workbench CAD mask for the 90nm transistor gate mask. The comparison was made in terms of accuracy and speed. Edge placement error (EPE) and intersection over union (IOU) were used as metrics in the work. The use of neural networks made it possible to speed up the calculation of the mask by 100 times while maintaining an accuracy of 92% on the IOU metric.

Keywords: computational photolithography, inverse lithography technology, convolutional neural networks, ML-OPC, U-Net, Erf-Net, Deep Lab v3, image-to-image, electronic design automation, artificial intelligence, edge placement error, intersection over union

Литература

1. Chien P., Chen M. Proximity effects in submicron optical lithography //Optical microlithography VI. – International Society for Optics and Photonics, 1987. – Т. 772. – С. 35-41.

2. Balasinski A. et al. A novel approach to simulate the effect of optical proximity on MOSFET parametric yield //International Electron Devices Meeting 1999. Technical Digest (Cat. No. 99CH36318). – IEEE, 1999. – С. 913-916.

3. Wong A. K. K. Resolution enhancement techniques in optical lithography. – SPIE press, 2001. – Т. 47.

4. Балан Н. Н. и др. Основные подходы к моделированию формирования фоторезистивной маски в вычислительной литографии //Известия высших учебных заведений. Материалы электронной техники. – 2020. – Т. 22. – №. 4. – С. 279-289.

5. Otto Oberdan W, Garofalo Joseph G, Low KK et al. Automated optical proximity correction: a rules-based approach // Optical/Laser Microlithography VII / International Society for Optics and Photonics. — Vol. 2197. — 1994. — P. 278–293.

6. Li J. et al. Model-based optical proximity correction including effects of photoresist processes //Optical Microlithography X. – International Society for Optics and Photonics, 1997. – Т. 3051. – С. 643-651.

7. Hung C. Y. et al. First 65nm tape-out using inverse lithography technology (ILT) //25th Annual BACUS Symposium on Photomask Technology. – International Society for Optics and Photonics, 2005. – Т. 5992. – С. 59921U.

8. Красников Г. Я., Синюков Д. В. Проблемы и перспективы развития методов коррекции оптической близости для современных уровней технологии //Труды научного совета РАН" Фундаментальные проблемы элементной базы информационно-вычислительных и управляющих систем и материалов для ее создания". – 2019. – Т. 1. – №. 3. – С. 17.

9. Spence C., Goad S. Computational requirements for OPC //Design for Manufacturability through Design-Process Integration III. – International Society for Optics and Photonics, 2009. – Т. 7275. – С. 72750U.

10. Choi S., Shim S., Shin Y. Machine learning (ML)-guided OPC using basis functions of polar Fourier transform //Optical Microlithography XXIX. – International Society for Optics and Photonics, 2016. – Т. 9780. – С. 97800H.

11. Choi S., Shim S., Shin Y. Neural network classifier-based OPC with imbalanced training data //IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2018. – Т. 38. – №. 5. – С. 938-948.

12. Shi B. et al. Fast OPC repair flow based on machine learning //Design-Process-Technology Co-optimization for Manufacturability XIV. – International Society for Optics and Photonics, 2020. – Т. 11328. – С. 113281B.

13. Shin Y. Computational Lithography Using Machine Learning Models //IPSJ Transactions on System LSI Design Methodology. – 2021. – Т. 14. – С. 2-10.

14.Tryasoguzov, P.E., Kuzovkov, A.V., Karandashev, I.M. et al. Using Machine Learning Methods to Predict the Magnitude and the Direction of Mask Fragments Displacement in Optical Proximity Correction (OPC). Opt. Mem. Neural Networks 30, 291–297 (2021). https://doi.org/10.3103/S1060992X21040056

15. Ye W. et al. LithoGAN: End-to-end lithography modeling with generative adversarial networks //2019 56th ACM/IEEE Design Automation Conference (DAC). – IEEE, 2019. – С. 1-6.

16. Yang H. et al. GAN-OPC: Mask optimization with lithography-guided generative adversarial nets //IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2019. – Т. 39. – №. 10. – С. 2822-2834.

17. Sun X. et al. U-Net convolutional neural network-based modification method for precise fabrication of three-dimensional microstructures using laser direct writing lithography //Optics Express. – 2021. – Т. 29. – №. 4. – С. 6236-6247.

18. Shao H. C. et al. From IC Layout to Die Photograph: A CNN-Based Data-Driven Approach //IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2020. – Т. 40. – №. 5. – С. 957-970.

19. Ma X. et al. Model-driven convolution neural network for inverse lithography //Optics express. – 2018. – Т. 26. – №. 25. – С. 32565-32584.

20. Ma X., Zheng X., Arce G. R. Fast inverse lithography based on dual-channel model-driven deep learning //Optics Express. – 2020. – Т. 28. – №. 14. – С. 20404-20421.

21. Ronneberger O., Fischer P., Brox T. U-net: Convolutional networks for biomedical image segmentation //International Conference on Medical image computing and computer-assisted intervention. – Springer, Cham, 2015. – С. 234-241.

22. Romera E. et al. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation //IEEE Transactions on Intelligent Transportation Systems. – 2017. – Т. 19. – №. 1. – С. 263-272.

23. Chen L. C. et al. Encoder-decoder with atrous separable convolution for semantic image segmentation //Proceedings of the European conference on computer vision (ECCV). – 2018. – С. 801-818.

24. Google Colab, https://colab.research.google.com

25. Pytorch Documentation, https://pytorch.org/docs/stable/index.html

26. Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In International Conference on Medical image computing and computer-assisted intervention, pp. 234-241. Springer, Cham, 2015.

27. Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", in IEEE International Conference on Computer Vision (ICCV), 2017

28. “Algorithm and methodology for increasing the OPC-recipe efficiency” Medvedev Konstantin A., Kuzovkov Alexey V., Ivanov Vladimir V. 10.22184/NanoRus.2019.12.89.368.372

Теплопроводность тонких кремниевых эллиптических наноструктур

Н.В. Масальский[1]

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

Аннотация. Обсуждается поведение аномальной теплопроводности кремниевых цилиндрических наноструктур с эллиптическим сечением в стационарных состояниях. При помощи численного моделирования показано, что аномальная теплопроводность связана с формой поперечного сечения. Чем меньше отношение меньшей полуоси эллипса к большей, тем выше теплопроводность эллиптических наноструктур. А также при повышении температуры вклад в аномальную теплопроводность из-за снижения числа Кнудсена превалирует над вкладом из-за снижения объемной теплопроводности. Поэтому, с ростом температуры наблюдается рост теплопроводности цилиндрических наноструктур с эллиптическим сечением. Полученные результаты демонстрируют аномальную природу теплопроводности тонких кремневых цилиндрических наноструктур с эллиптическим сечением, которая отличается от классических представлений о теплопроводности твердых тел. аннотации.

Ключевые слова: теплопроводность, число Кнудсена, уравнение Максвелла-Каттанео-Вернотта, кремниевые наноструктуры

1. Введение

Аномальная природа тепловых процессов в низко размерных структурах связана с влиянием отношения между «средним свободным пробегом» теплоносителей (l_{ph}) и соответствующим характерным размером низко-размерной системы [1]. Опираясь на обширные исследования, можно прогнозировать, что теплопроводность нанопровода зависит от множества факторов, таких как материал, размер поперечного сечения, формы и технологии изготовления [2-5]. В настоящей работе мы исследуем случай, при котором поперечное сечение нанопроволоки отличное от круглого, является эллиптическим, что обусловлено параметрическим разбросом параметров техпроцесса [6].

Цель работы – с помощью фононной гидродинамики численно исследовать аномальную теплопроводность кремниевых цилиндрических наноструктур с эллиптическим сечением в стационарных состояниях. При этом тепловые характеристики материала учитываются посредством его объемной теплопроводности.

2. Моделирование теплопроводности в стационарном состоянии

В модельном представлении поперечное сечение нанопроволки есть эллипс где 2a – большая ось и 2b – малая ось (см. рис. 1), длина нанопроволки L значительно больше 2a, и направление теплового потока совпадает с положительным направлением оси z. В приближении, что отношение между характеристическим размером системы и средним свободным пробегом (т. е. число Кнудсена Kn) всегда больше единицы ($\mathrm{Kn}>1$), то теплоносители переходят в гидродинамический режим [4, 5]. Следует отметить, что приложениях найти характерный размер системы не так просто, как может показаться [7]. Неправильный выбор этого параметра (точнее его численное значение) может привести к неверным результатам [8]. Здесь мы предполагаем, что $\mathrm{Kn}=\mathrm{l}_{\mathrm{ph}}/\mathrm{b}$ потому что преобладающими столкновениями фононов со стенками будут те, которые соответствуют меньшему размеру. В этом случае решение общего уравнения Максвелла-Каттанео-Вернотта (МКВ) [9] для объемного теплового потока (т. е. для пропадающего теплового потока у стенок) имеет вид [10, 11]

$$\mathrm{q_v(x,y)}=\frac{k(T)}{2Kn(T)^2}\left(\frac{1}{1+\gamma^2}\right)\left(1-\frac{x^2}{a^2}-\frac{y^2}{b^2}\right)\frac{\Delta T}{L}, \qquad (1)$$

где $\gamma=b/a$.

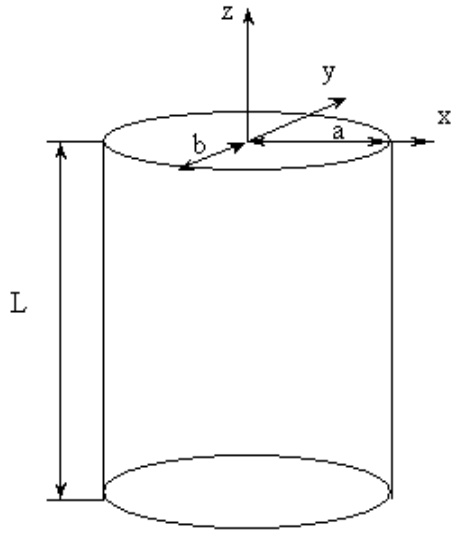

Рис. 1. Нанопроволока с эллиптическим поперечным сечением, где a – большая полуось (совпадает с осью x), b – малая полуось (совпадает с осью y), L – длина нанопроволоки, ось z – направление распространения теплового потока

Из (1), легко получить, что вклад теплового потока на границе в виде:

$$q_b(x) = \frac{k(T)}{Kn(T)}\left(\frac{C}{1+\gamma^2}\right) \frac{\Delta T}{\sqrt{\frac{x^2}{a^2}(\gamma^2-1)+1}\,L}, \qquad (2)$$

Следует отметить, что, даже при $l_{ph} >> b$, поток qb нельзя рассматривать как постоянную величину по всему поперечному сечению, поскольку нормальная производная не всегда имеет одинаковое значение вдоль эллиптической границы.

Уравнения (1) и (2) позволяют получить следующее выражение для аномальной теплопроводности [11]:

$$k_e(Kn) = \frac{\int_{-b}^{b} dy \int_{-\sqrt{1-y^2/b^2}}^{\sqrt{1-y^2/b^2}} dx[q_v(x,y)+q_b(x)]}{\pi ab\left(\frac{\Delta T}{L}\right)} =$$

$$= \frac{k(T)}{4Kn(T)}\left(\frac{1}{Kn(T)}\left(\frac{1}{1+\gamma^2}\right)+ + \frac{C}{2}\left(1-0.6976\left(\frac{\gamma^2-1}{1.915\gamma^2+1}\right)\right)\right) \qquad (3)$$

Выражение (3) позволяет исследовать поведение теплопроводности цилиндрических наноструктур с эллиптическим сечением в зависимости от топологических параметров и температуры, не прибегая к сложным вычислениям.

3. Результаты моделирования

Ниже на рис. 2 приведены результаты численного моделирования аномальной теплопроводности в эллиптических кремниевых нанопроводах, которые получены из (3). Модельное поведение исследовано относительно параметров Kn и γ.

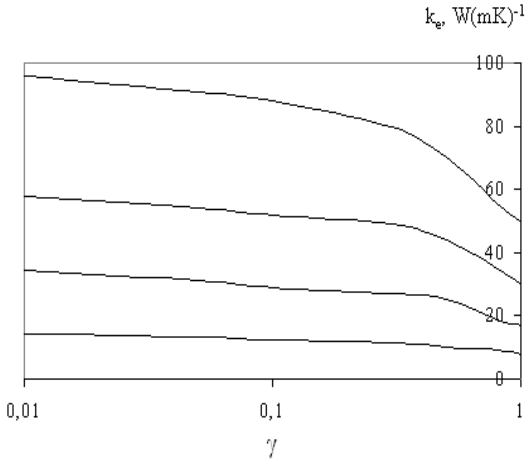

Рис. 2. Зависимость ke от γ при разных Kn, где Kn=2 – верхняя кривая, ниже Kn = 3, 5 и 10. Значения γ представлены в логарифмическом масштабе.луось (совпадает с осью y), L – длина нанопроволоки, ось z – направление распространения теплового потока

Из результатов, приведенных на рис. 2 следует, что аномальная теплопроводность связана с формой поперечного сечения. В частности, теоретическая модель показывает, что чем меньше γ, тем выше теплопроводность эллиптических наноструктур. Следует отметить, что примерно в диапазоне γ от 1 до 0.2 эта зависимость степенная, а при меньших значениях γ она становиться практически линейной для любого Kn, т.е. когда нанотрубка становится более похожей на пленку.

Мы отметили два придельных случая γ =1 круглого сечения и γ =0,01 сильно сплющенного эллипса. Результаты расчетов приведены на рис. 3.

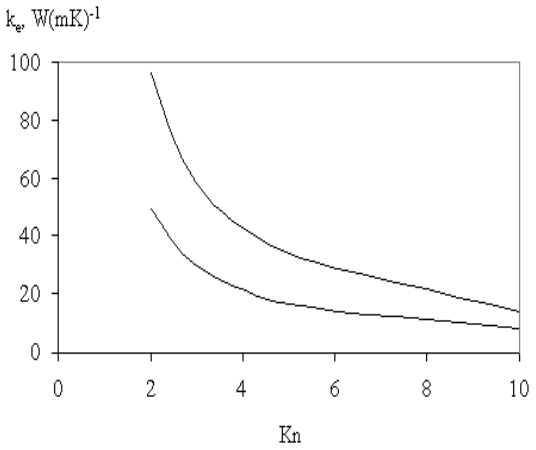

Рис. 3. Зависимость ке от Kn для γ =0, 01 (верхняя кривая) и γ=1 (нижняя кривая)

Отметим, что для любого Kn отношение

$$\frac{k_e(\gamma = 0.01)}{k_e(\gamma = 1)} \approx 2$$

Сопоставим значения теплопроводности эллиптических волокон и пленок с идеальным прямоугольным сечением. При условии $W >> h$, где W, h - ширина и высота пленки, соответственно, то можно получить следующее выражение для аномальной теплопроводности. Пренебрегая членами, содержащими масштабирующий коэффициент второго порядка и выше, решения МКВ сводится к [10, 11]:

$$k_1(Kn) = \frac{k(T)}{Kn(T)} \frac{1}{12Kn(T)}(1 - 0.63\chi + 6CKn(T)(1 - 0.271\chi)), \quad (4)$$

В [11] утверждается, что приближенное решение (4) практически не отличается от точного. Тогда, для $\gamma = \chi = 0,01$ значение ке превышает $к_1$ на 3.5% для b=h=10 нм, при $\gamma = \chi = 0,1$ отличие составляет 9.6%. Поэтому значение аномальной теплопроводности эллиптических наноструктур может служить первоначальным приближением для оценки теплопроводности неидеальной прямоугольной пленки.

Исследование температурной зависимости теплопроводности представлено на ниже рис. 4 и 5. Хорошо известно, что с повышением температуры объемная теплопроводность кремния снижается. Аналогичным образом ведет себя и число Кнудсена. Данная зависимость в диапазоне температур 200…400 К представлена на рис. 4.

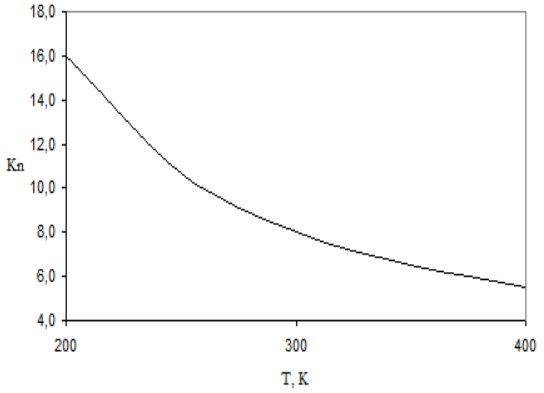

Рис. 4. Зависимость Kn от T для b=10 нм

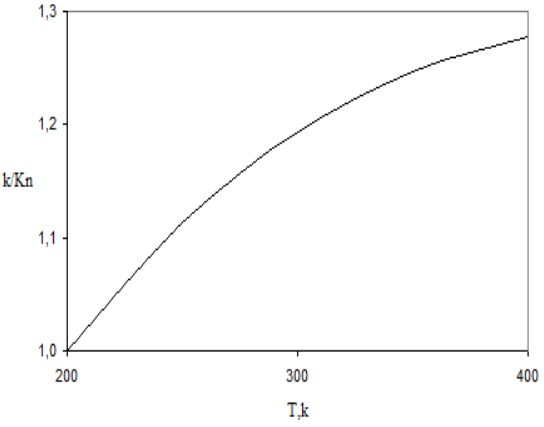

Рис. 5. Зависимость к/Kn от T

Переход к другим размерам можно реализовать умножив значение Kn при выбранной температуре на отношение т.е. $Kn = 0.1b_c Kn(b = 10)$

Сопоставив два выражения (3) и (4), можно сделать вывод, что они включают отношение $k(T) / Kn(T)$, которое существенным образом определяет температурную зависимость теплопроводности и пленок и нитей. Выше было отмечено, что, во-первых, с уменьшением Kn теплопроводность возрастает, и, во-вторых, с ростом температуры объемная теплопроводность падает. В результате с повышением температуры задействованы два взаимно подавляющих механизма. Этот процесс можно интерпретировать в виде отношения $k(T) / Kn(T)$. На рис 5 представлена температурная зависимость данного отношения. Поскольку сравниваемые величины имеют разные размерности, мы первоначально их нормируем относительно значений при T=200 К. Затем нормированные значения используем для вычисления соотношения $k(T) / Kn(T)$. Из результатов расчетов видно, что вклад в аномальную теплопроводность из-за снижения Kn превалирует над вкладом из-за

снижением к. И, следовательно, с ростом температуры наблюдается рост теплопроводности цилиндрических наноструктур с эллиптическим сечением. Примерно на 28% в диапазоне от 200 до 400 К.

4. Заключение

На основе фононной гидродинамики с использованием уравнения Максвелла-Каттанео-Вернотта численно исследовано поведение аномальной теплопроводности кремниевых цилиндрических эллиптических наноструктур с эллиптическим сечением в стационарных состояниях. Показано, что аномальная теплопроводность связана с формой поперечного сечения. Чем меньше отношение меньшей полуоси эллипса к большей, тем выше теплопроводность эллиптических наноструктур. При этом в диапазоне отношений от 1 до 0.2 эта зависимость степенная, а при меньших значениях она становиться практически линейной для любого числа Кнудсена, т.е. когда наноструктура становится более похожей на пленку.

Исследована температурная зависимость аномальной теплопроводности. Показано, что при повышении температуры вклад в аномальную теплопроводность из-за снижения числа Кнудсена превалирует над вкладом из-за снижения объемной теплопроводности. Следовательно, с ростом температуры наблюдается рост теплопроводности цилиндрических наноструктур с эллиптическим сечением на 28% в диапазоне от 200 до 400 К.

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".

Thermal Conductivity of Thin Silicon Elliptical Nanostructures

N. Masalsky

Abstract. The behavior of anomalous thermal conductivity of silicon cylindrical elliptical nanostructures with elliptical cross-section in stationary states is discussed. Numerical simulation has shown that the abnormal thermal conductivity is related to the shape of the cross-section. The smaller the ratio of the large semi-axis of the ellipse to the smaller one, the higher the thermal conductivity of elliptical nanostructures. And that with an increase in temperature, the contribution to abnormal thermal conductivity due to a decrease in the Knudsen number prevails over the contribution due to a decrease in volumetric thermal conductivity. Therefore, with an increase in temperature, the thermal conductivity of cylindrical nanostructures with an elliptical cross-section increases by 28% in the range from 200 to 400 K. The obtained results demonstrate the anomalous nature of the thermal conductivity of thin silicon cylindrical nanostructures with an elliptical cross-section, which distinguishes them from the classical ideas about the thermal conductivity of solids.

Keywords: thermal conductivity, Knudsen number, Maxwell-Cattaneo-Vernott equation, silicon nanostructures

Литература

1. J.H. Davies. The physics of low dimensional semiconductors. New York, Plenum, 1998.

2. R. Luzzi, A.R. Vasconcellos, J. Galvão Ramos. Predictive statistical mechanics: A non equilibrium ensemble formalism. Fundamental theories of physics. Dordrecht, Kluwer, 2002.

3. W. Liu, M. Asheghi. Phonon-boundary scattering in ultrathin single-crystal silicon layers. “Appl. Phys. Lett.”, V 84, (2004), 3819–3821.

4. Z.M. Zhang. Nano/Microscale heat transfer. New York, McGraw-Hill, 2007.

5. D. Jou, J. Casas-Vázquez, G. Lebon. Extended irreversible thermodynamics, 4th revised edn. Berlin, Springer, 2010.

6. F. Schwierz, H.Wong, J. J. Liou. Nanometer CMOS. Singapore, Pan Stanford Publishing, 2010.

7. A. Sellitto, V. Cimmelli, D. Jou. Mesoscopic theories of heat transport in nanosystems. Springer Cham Heidelberg New York Dordrecht London, Springer International Publishing Switzerland, 2016.

8. Н.В. Масальский. Аномальная теплопроводность в кремниевых цилиндрических наноструктурах. “Труды НИИСИ”, Т 10, (2020), 45-51.

9. C. Cattaneo. Sur une forme de l'équation de la chaleur éliminant le paradoxe d'une propagation instantanée. "C. R. Acad. Sc.", V. 247, (1958), 431–433.

10. F.X. Alvarez, V.A. Cimmelli, D. Jou, A. Sellitto. Mesoscopic description of boundary effects in nanoscale heat transport. "Nanoscale Systems MMTA", V. 1, (2012), 112–142.

11. A. Sellitto, F.X. Alvarez, D. Jou. Geometrical dependence of thermal conductivity in elliptical and rectangular nanowires. "Int. J. Heat Mass Transfer", V. 55, (2012), 3114–3120.

Подписано в печать 16.12.2022 г.
Формат 60x90/8
Печать цифровая. Печатных листов 10.75
Тираж 100 экз. Заказ № 1289

Отпечатано в ФГБУ «Издательство «Наука»
121099, Москва, Шубинский пер., 6