DOI: 10.15514/ISPRAS-2024-36(5)-3

Открытое промежуточное представление специализированных потоковых вычислителей, основанное на MLIR

[1,2] *А.С. Камкин, ORCID: 0000-0001-6374-8575 <kamkin@ispras.ru>*
[2] *М.Ю. Литвинов, ORCID: 0009-0009-8926-139X <litvinov@ispras.ru>*
[2] *И.А. Григоров, ORCID: 0009-0001-6373-738X <grigorovia@ispras.ru>*

[1] *Российский экономический университет имени Г.В. Плеханова, 117997, Россия, г. Москва, Стремянный пер., д. 36.*
[2] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*

Аннотация. В последнее время для решения вычислительных задач с жесткими ограничениями на производительность (пропускную способность) и энергопотребление (потребляемую мощность) широко используются гетерогенные компьютерные системы. Как правило, такие системы состоят из микропроцессоров общего назначения и аппаратных ускорителей на базе ПЛИС, реализующих наиболее затратные операции (обычно отражающих специфику предметной области). Данная статья посвящена автоматизации проектирования аппаратных ускорителей, ориентированных на задачи потоковой обработки данных (streaming data computing). Особенностями ускорителей этого типа (и решаемых ими задач) являются: (1) непрерывные (на каждом такте работы) прием и выдача данных; (2) ограниченная (по времени и памяти) зависимость выходных данных от входных. Потоковая обработка охватывает широкий класс приложений, включая цифровую обработку сигналов, шифрование трафика, численное моделирование, биоинформатику и другие. В работе предлагается концепция языка DFCIR (DataFlow Computer Intermediate Representation), предназначенного для промежуточного представления моделей потоковых вычислителей. Язык DFCIR основан на открытой компиляторной инфраструктуре MLIR (Multi-Level Intermediate Representation). Для построения RTL-моделей ускорителей по DFCIR-описаниям используются средства CIRCT (Circuit IR Compilers and Tools) – подпроекта MLIR, объединяющего инструменты для работы с моделями аппаратуры.

Ключевые слова: гетерогенная компьютерная система; аппаратный ускоритель; специализированный потоковый вычислитель; программируемая логическая интегральная схема; автоматизация проектирования аппаратуры; язык промежуточного представления; инфраструктура MLIR; инструментарий CIRCT.

Для цитирования: Камкин А.С., Литвинов М.Ю., Григоров И.А. Открытое промежуточное представление специализированных потоковых вычислителей, основанное на MLIR. Труды ИСП РАН, 2024, том 36 вып. 5, с. 31-46. DOI: 10.15514/ISPRAS-2024-36(5)-3.

Благодарности: Исследование выполнено в РЭУ им. Г.В. Плеханова за счет гранта Российского научного фонда № 23-21-00313, https://rscf.ru/project/23-21-00313/. От лица ИСП РАН (участника университетской программы Maxeler MAX-UP) авторы благодарят компанию Maxeler Technologies за предоставление доступа к инструменту MaxCompiler.

Open-Source MLIR-Based Intermediate Representation for Application-Specific Streaming Computers

[1,2] *A.S. Kamkin, ORCID: 0000-0001-6374-8575 <kamkin@ispras.ru>*
[2] *M.Yu. Litvinov, ORCID: 0009-0009-8926-139X <litvinov@ispras.ru>*
[2] *I.A. Grigorov, ORCID: 0009-0001-6373-738X <grigorovia@ispras.ru>*
[1] *Plekhanov Russian University of Economics,*
36 Stremyanny lane, Moscow, 117997, Russia.
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25 Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. Recently, heterogeneous computer systems have been widely used to solve computational tasks with strict constraints on performance (throughput) and power consumption. Typically, such systems consist of general-purpose microprocessors and FPGA-based hardware accelerators implementing the most expensive operations (which are usually application-specific ones). This article is devoted to the design automation of hardware accelerators for streaming data computing. The features of this type of accelerators (and the problems they solve) are as follows: (1) continuous (cycle-by-cycle) reception and production of data; (2) bounded (in time and memory) output-input dependence. Streaming data computing covers a wide range of applications, including digital signal processing, traffic encryption, numerical modeling, bioinformatics, etc. The paper introduces the concept of DFCIR (DataFlow Computer Intermediate Representation), a language for an intermediate representation of streaming data computing designs. The DFCIR language is based on the open compiler infrastructure MLIR (Multi-Level Intermediate Representation). RTL models of accelerators are built from DFCIR descriptions with the use of CIRCT (Circuit IR Compilers and Tools), a subproject of MLIR that combines tools for working with hardware designs.

Keywords: heterogeneous computer system; hardware accelerator; application-specific streaming computer; field-programmable gate array; electronic design automation; intermediate representation language; MLIR; CIRCT.

For citation: Kamkin A.S., Litvinov M.Yu., Grigorov I.A. Open-Source MLIR-Based Intermediate Representation for Application-Specific Streaming Computers. Trudy ISP RAN/Proc. ISP RAS, vol. 36, issue 5, 2024. pp. 15-19 (in Russian). DOI: 10.15514/ISPRAS-2024-36(5)-3.

Acknowledgements: This research has been completed in Plekhanov Russian University of Economics and funded by Russian Science Foundation (grant № 23-21-00313), https://rscf.ru/project/23-21-00313/. On behalf of ISP RAS (a member of the Maxeler University Program MAX-UP), the authors thank Maxeler Technologies for providing access to MaxCompiler.

1. Введение

Для решения вычислительных задач при наличии жестких ограничений на производительность (пропускную способность) и энергопотребление (потребляемую мощность) широко используются гетерогенные компьютерные системы (ГКС), то есть системы, состоящие из разнородных вычислительных элементов [1, 2]. Типичная ГКС содержит микропроцессор общего назначения и несколько аппаратных ускорителей, обычно на базе ПЛИС, реализующих наиболее затратные операции (в зависимости от предметной области это могут быть матричные умножения, криптографические примитивы и т.п.). За более высокую эффективность ГКС приходится расплачиваться усложнением процессов разработки и верификации: создание ускорителей требует привлечения языков описания аппаратуры (hardware description languages, HDL) [3-5]; для интеграции вычислительных элементов нужна специализированная операционная среда (runtime); отладка предполагает использование средств ко-эмуляции программных и аппаратных компонентов. Таким образом, актуальной задачей является разработка технологий автоматизации проектирования как ГКС в целом, так и отдельных аппаратных ускорителей, применяемых в ГКС.

В работе рассматриваются аппаратные ускорители определенного вида, а именно специализированные вычислители, ориентированные на задачи потоковой обработки данных (application-specific streaming computers); далее в статье вычислители этого вида называются потоковыми вычислителями (ПВ). Потоковая обработка (streaming computing) имеет много приложений: цифровая обработка сигналов, шифрование трафика, численное моделирование, биоинформатика и многие другие. Можно выделить две основные особенности ПВ (и решаемых ими задач): (1) непрерывные (на каждом такте работы) прием и выдача данных; (2) ограниченная (по времени и памяти) зависимость выходных данных от входных. Структурно ПВ представляют собой параллельно-конвейерные устройства с большим числом стадий (иногда более 1000), что роднит их с систолическими массивами [6], однако в отличие от последних ПВ не обязательно имеют однородную структуру. Большая пропускная способность ПВ обеспечивается высокой частотой работы, что достигается специальными методами планирования вычислений (scheduling) [7] и оптимизации временных характеристик (retiming) [8].

Ключевым элементом технологии проектирования аппаратных ускорителей является язык промежуточного представления (intermediate representation, IR): помимо возможностей, которые язык дает для описания функциональности и структуры вычислителя, он так или иначе определяет набор трансформаций, которые можно совершать над описанием (моделью). В работе предлагается язык DFCIR (DataFlow Computer Intermediate Representation), нацеленный на описание специализированных ПВ. В основе данного языка лежит открытая компиляторная инфраструктура MLIR (Multi-Level Intermediate Representation) [9], разрабатываемая командой LLVM [10]. MLIR позволяет создавать специализированные языки (диалекты) и средства трансформации, позволяющие переходить от описания на одном диалекте к описанию на другом, в том числе понижать уровень абстракции описания (lowering). С использованием разработанных средств проектирование ПВ организуется как последовательность преобразований MLIR-представлений: (1) предметно-ориентированное представление (выходит за рамки работы); (2) DFCIR-описание (оптимизированное под производительность или энергопотребление); (3) представление на базе диалектов CIRCT (Circuit IR Compilers and Tools) [11]; (4) HDL-описание (результат обработки CIRCT-представления). Важно отметить, что инфраструктура MLIR дает возможность разрабатывать программные и аппаратные компоненты ГКС в рамках единой среды проектирования.

В статье представлены следующие результаты: (1) требования к языку промежуточного представления специализированных ПВ и средства для их реализации; (2) язык DFCIR промежуточного представления специализированных ПВ, основанный на компиляторной инфраструктуре MLIR; (3) средства трансляции из DFCIR в языки описания аппаратуры Verilog/SystemVerilog, основанные на инструментарии CIRCT.

Статья организована следующим образом. Раздел 2 содержит базовые сведения о проектировании цифровой аппаратуры и специализированных ПВ. В разделе 3 рассматриваются существующие разработки (языки и инструменты), используемые для проектирования ПВ. Раздел 4 посвящен предлагаемому языку DFCIR: описываются типы данных и конструкции языка; приводятся примеры описаний; определяется схема трансляции в FIRRTL (один из диалектов CIRCT). В разделе 5 приводятся результаты экспериментального анализа разработанных средств. Раздел 6 является заключением; в нем подводятся итоги работы и определяются направления дальнейших исследований.

2. Предметная область

Традиционный маршрут проектирования цифровой аппаратуры начинается с создания модели уровня регистровых передач (register-transfer level, RTL). RTL-модель описывает аппаратуру в терминах пересылок данных между регистрами (элементами памяти) и

функциональными элементами (выполняющими арифметико-логические преобразования данных). Для разработки RTL-моделей используют языки описания аппаратуры, наиболее распространенными из которых являются Verilog [3], SystemVerilog [4] и VHDL [5].

Разработка аппаратуры на уровне регистровых передач – это трудозатратный процесс, при этом изменение требований к целевым характеристикам устройства (частоте, площади и/или энергопотреблению) может потребовать существенной переработки RTL-модели (что чревато внесением ошибок). Актуальной задачей является создание средств автоматизированной разработки аппаратуры.

2.1. Подходы к автоматизации проектирования

Автоматизация создания RTL-моделей обеспечивается двумя основными способами [12]:

1. параметризация – вместо одной модели, создается семейство моделей, рассчитанных на разные случаи (режимы) работы; достижение целевых характеристик реализуется подбором значений параметров;
2. повышение уровня абстракции – при описании аппаратуры игнорируются детали реализации, упор делается на поведении; структура устройства конкретизируется в результате планирования вычислений.

В первом случае используются как стандартные HDL-языки (в современных редакциях этих языков есть средства параметризации и макрогенерации), так и специализированные языки, предназначенные для конструирования аппаратуры (hardware construction): Chisel [13], MyHDL [14], Bluespec SystemVerilog [15] и другие. Достоинством подхода является предсказуемый результат: сгенерированные модели по характеристикам сопоставимы с разработанными вручную. Недостаток – недостаточно высокий уровень автоматизации.

Во втором способе, известном как высокоуровневый синтез (high-level synthesis, HLS) [16], часто используются популярные языки программирования: C, C++ и другие. Примерами HLS-инструментов являются Vitis HLS [17], Bambu [18] и LegUp [19]. Следует отметить, что фон-неймановская модель вычислений, лежащая в основе императивных языков программирования, плохо согласуется с «неограниченным» параллелизмом аппаратуры, что затрудняет достижение высоких характеристик синтезируемых схем. Лучшие результаты достигаются при использовании языков, базирующихся на функциональных и потоковых парадигмах [12].

Предлагаемый подход занимает промежуточную нишу между конструированием аппаратуры и высокоуровневым синтезом: с одной стороны, он обеспечивает контролируемый результат синтеза; с другой стороны, позволяет абстрагироваться от некоторых деталей аппаратуры (прежде всего связанных с временны́ми характеристиками: задержкой устройства, периодичностью поступления данных и т.п.).

2.2. Потоковые вычислители

Настоящая работа фокусируется на синхронных статически конфигурируемых потоковых вычислителях, далее для краткости называемых просто потоковыми вычислителями (ПВ). Вычислители этого типа принимают и выдают данные непрерывно, на каждом такте работы; подразумевается, что зависимость выходных данных от входных ограничена по времени и требует для своего выражения небольшого объема памяти. ПВ можно представить в виде графа, узлами которого являются порты ввода/вывода, функциональные элементы (ФЭ) и блоки памяти, а дугами – каналы передачи данных. Все компоненты ПВ функционируют синхронно, при этом ФЭ являются конвейерными устройствами без блокировок (глубины конвейеров разных ФЭ могут различаться, но для каждого ФЭ это число фиксировано и известно на этапе планирования вычислений). При синтезе ПВ для выравнивания времен поступления данных по разным каналам на один ФЭ в схему вставляются вспомогательные

очереди (FIFO); также могут выполняться оптимизирующие преобразования, меняющие структуру вычислителя (изложение этого вопроса выходит за рамки статьи).

На рис. 1 приведен пример ПВ для вычисления скалярного произведения двух трехэлементных векторов. Вычислитель имеет шесть портов ввода и один порт вывода: x_1, x_2, x_3 – координаты первого вектора; y_1, y_2, y_3 – координаты второго вектора; z – результат. Предположим, что выполнение операции умножения занимает 5 тактов, а выполнение операции сложения – 2 такта. Для выравнивания времен поступления данных на сумматор **6** со стороны умножителя **3** вставляется двухэлементная очередь **5**. Для данного ПВ задержка вычислений (latency) составляет 9 тактов (сначала за 5 тактов параллельно выполняются три умножения, затем последовательно выполняются два сложения, каждое из которых занимает 2 такта); периодичность (periodicity) входных и выходных потоков (время между двумя последовательными пересылками операндов и результатов) – 1 такт; соответственно, производительность (throughput) – 1 операция за такт (при частоте 100 МГц – 100 млн операций в секунду).

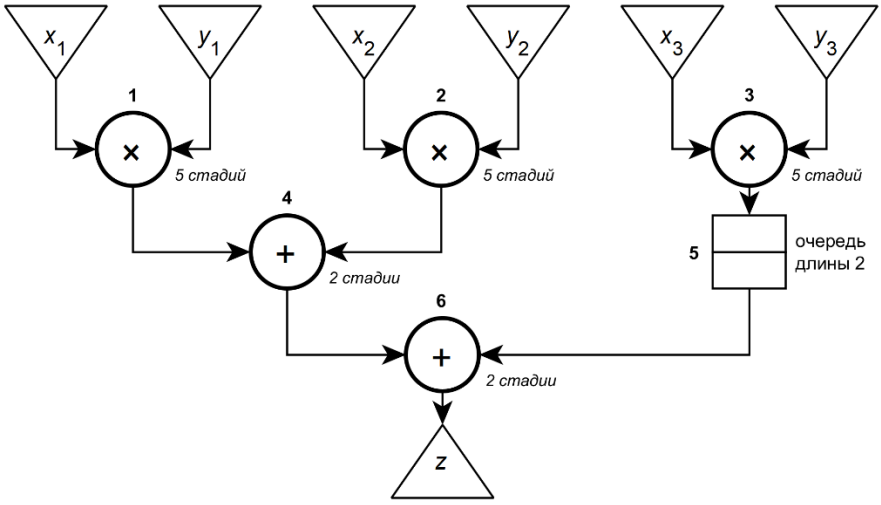

Рис. 1. Потоковый вычислитель, реализующий скалярное произведение векторов (вариант 1).
Fig. 1. Streaming computer implementing the dot product (case 1).

На рис. 2 показан другой ПВ, также вычисляющий скалярное произведение двух трехэлементных векторов. Данный ПВ имеет два порта ввода и один порт вывода: x_i и y_i – координаты первого и второго вектора соответственно (координаты подаются последовательно); z – результат. По сравнению с ПВ из первого примера здесь используется один умножитель вместо трех. Для определения границ векторов в потоках координат используется устройство **3** – счетчик по модулю 3. Компаратор **4** проверяет, что значение счетчика не превосходит 1; результат сравнения (один бит) подается в трехэлементную очередь **5**, выход которой соединяется с селектором мультиплексора **6**. Заметим, что очередь в данном случае реализована в форме сдвигового регистра; его разрядность (L_5) согласована с числом стадий выполнения операций умножения (L_1), сравнения (L_4) и мультиплексирования (L_6): $L_1 = L_4 + L_5 + L_6$. Таким образом, результат умножения первой пары координат векторов будет складываться с нулем, как и результат умножения второй пары. Здесь требуется пояснение: выполнение сложения занимает 2 такта, поэтому, когда умножитель выдает произведение очередной пары координат, сумма произведений предшествующих пар еще не готова; возникает эффект «расслоения» потока – на одних позициях потока вычисляются суммы произведений пар координат с нечетными номерами, на других (с противоположной четностью) – наоборот, с нечетными номерами. Для

«склейки» двух образовавшихся «подпотоков» используется регистр **7** и сумматор **8**. Оставшиеся элементы представленной схемы обеспечивают стробирование выходного потока (они являются опциональными): всякий раз когда значение счетчика становится равным 2, единичный бит помещается в младший разряд сдвигового регистра **10**, старший бит которого используется в качестве строба. Разрядность регистра (L_{10}) определяется условием: $L_9 + L_{10} = L_1 + L_2 + L_8$.

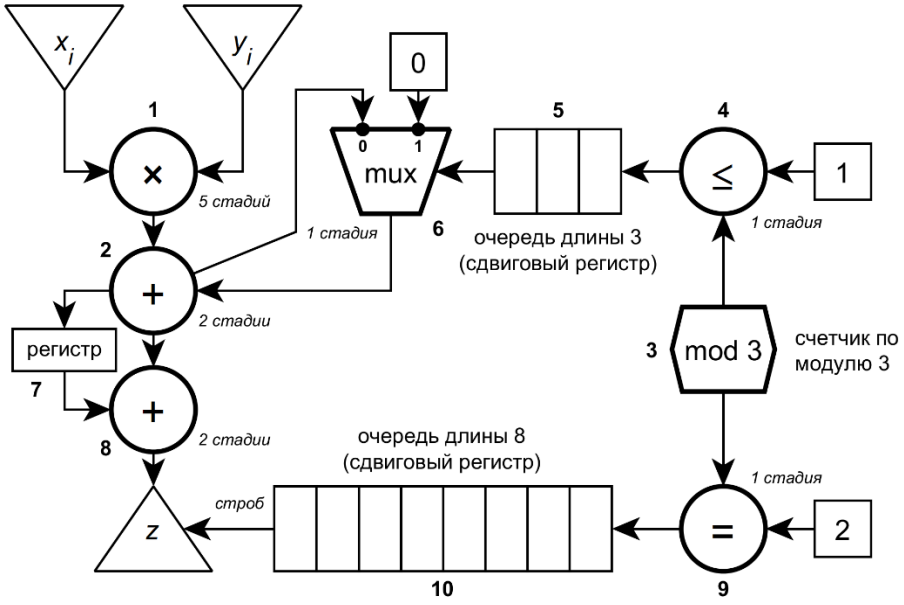

Рис. 2. Потоковый вычислитель, реализующий скалярное произведение векторов (вариант 2).
Fig. 2. Streaming computer implementing the dot product (case 2).

2.3. Требования к языку промежуточного представления

Центральным элементом технологии автоматизированного проектирования аппаратуры (конструирования или высокоуровневого синтеза) является язык промежуточного представления (ЯПП). ЯПП имеет ту же выразительную мощь, что и исходный язык, но при этом лишен избыточности («синтаксического сахара»), характерной для человеко-ориентированных языков. Минималистичность ЯПП упрощает формализацию его семантики и реализацию трансформаций.

К ЯПП ПВ были сформулированы следующие требования:

1. ориентация на потоковую обработку данных (наличие в языке такой сущности, как поток, и операций для работы с потоками);
2. возможность абстрагирования от временны́х характеристик (задержки, периодичности) отдельных ФЭ и ПВ в целом;
3. наличие механизмов конкретизации временны́х характеристик ФЭ и ПВ (фиксирования результатов планирования вычислений);
4. бесшовная интеграция с ЯПП выше- и нижележащих уровней (например, на основе инфраструктуры MLIR).

3. Существующие разработки

В настоящее время имеется широкий спектр языков, применяемых (или потенциально применимых) в проектировании ПВ. Их можно разделить на языки высокоуровневого описания и ЯПП. Краткая характеристика этих языков приведена в табл. 1.

Табл. 1. Краткая характеристика языков, используемых в проектировании потоковых вычислителей.
Table 1. Brief description of the languages used in the design of streaming computers.

Язык	Лицензия	Специализация	Уровень абстракции	Тип языка
MaxJ	Коммерческая	Потоковые вычислители	Поведенческий уровень	Язык высокоуровневого описания
DSLX	Apache License v2.0	Потоковые вычислители	Поведенческий уровень	Язык высокоуровневого описания
Пифагор	*Нет данных*	Потоковые вычислители	Поведенческий уровень	Язык высокоуровневого описания
HIR	Apache License v2.0	Аппаратные ускорители	Поведенческий уровень	Язык промежуточного представления
FIRRTL	Apache License v2.0	Аппаратура общего назначения	Уровень регистровых передач	Язык промежуточного представления
HW/Comb/ Seq	Apache License v2.0	Аппаратура общего назначения	Уровень регистровых передач и вентилей	Язык промежуточного представления
LLHD	Apache License v2.0	Аппаратура общего назначения	Все уровни	Язык промежуточного представления

Будучи ориентированными на проектирование аппаратуры, данные языки имеют схожие системы типов: битовые векторы, кортежи, структуры, массивы; высокоуровневые языки поддерживают числа с фиксированной и плавающей точкой произвольной точности.

3.1 Языки высокоуровневого описания

В данном разделе представлена информация по языкам высокоуровневого описания ПВ: MaxJ (проект MaxCompiler) [20], DSLX (проект XLS) [21] и Пифагор [22]. Ключевой особенностью этих языков является абстрагирование (в той или иной мере) от временны́х характеристик проектируемой аппаратуры: компиляторы (синтезаторы) сами планируют вычисления с учетом заданных проектных ограничений и проводят разные оптимизации, в том числе конвейеризацию (pipelining) и ретайминг (retiming).

Язык MaxJ разработан в рамках проекта MaxCompiler компании Maxeler Technologies (в 2022 году приобретена компанией Groq) и предназначен для построения высокопроизводительных ПВ на базе ПЛИС. В отличие от других средств, рассматриваемых в статье, MaxCompiler позволяет создавать как отдельные аппаратные ускорители, так и ГКС в целом. В основе MaxJ лежит язык программирования Java, расширенный следующими абстракциями: поток (stream), скаляр (scalar), счетчик (counter), смещение в потоке (offset), локальная память (fast memory), глобальная память (large memory). Поток есть последовательность данных одного типа, а скаляр – фиксированное значение (параметр времени исполнения). Над потоками

определены стандартные арифметико-логические операции (сложение, вычитание, умножение, деление и другие): результатом является поток, элементы которого получаются применением заданной операции к соответствующим элементам потоков-операндов. Счетчики – это потоки специального вида (целочисленные прогрессии), используемые для организации циклов; вложенные циклы реализуются с помощью цепей счетчиков (counter chains). Смещения дают возможность обращаться к «прошлым» и «будущим» элементам потока (например, смещение -1 дает доступ к элементу, полученному такт назад). Блоки памяти позволяют сохранять промежуточные результаты вычислений.

Язык DSLX является входным языком открытого инструмента XLS, ориентированного на построение ПВ. Выходом XLS является SystemVerilog-описание, не привязанное к конкретным моделям ПЛИС и платам-ускорителям, что позволяет использовать инструмент в разработке блоков заказных интегральных схем. DSLX похож на язык программирования Rust, но в отличие от последнего учитывает специфику аппаратуры: объекты имеют фиксированную разрядность, графы вызовов функций допускают полную развертку и т.п. Следует отметить такие особенности языка, как параметрические структуры и функции, вывод типов (type inference) и конструкции for (циклы с фиксированным числом итераций).

Язык Пифагор, разрабатываемый в Сибирском федеральном университете, предназначен для создания переносимых параллельных программ и аппаратных реализаций ПВ. Язык базируется на функционально-потоковой парадигме: вычисления представляются информационным графом (отражающим ФЭ и каналы передачи данных); на основе информационного графа строится управляющий граф, отвечающий за синхронизацию работы ФЭ (формирование сигналов готовности результатов). Отличительной чертой компилятора с языка Пифагор является возможность редукции параллелизма, что может быть использовано для построения ускорителей на базе ПЛИС с небольшим числом логических элементов.

3.2 Языки промежуточного представления

В данном разделе даются краткие сведения по ЯПП, применимых к проектированию ПВ: HIR [23], FIRRTL [24], HW/Comb/Seq [25-27] и LLHD [28]. Все перечисленные языки реализованы в форме MLIR-диалектов и, за исключением HIR, развиваются в рамках зонтичного проекта CIRCT.

Язык HIR разработан в Индийском научном институте (Indian Institute of Science) и предназначен для представления аппаратных ускорителей общего назначения. Основными средствами описания вычислений в нем являются циклы, функции и конструкции доступа к памяти. Следует отметить поддержку многомерных тензоров, широко применяемых в задачах машинного обучения. В отличие от языков, рассматриваемых ниже, HIR не фиксирует потактовое поведение устройства, предоставляя свободу для планирования вычислений и оптимизации. Результаты планирования задаются с помощью так называемых временны́х переменных (time variables) и используются для генерации управляющих автоматов.

Язык FIRRTL (Flexible Intermediate Representation for Register Transfer Level) создавался в Калифорнийском университете в Беркли (UC Berkeley) как средство промежуточного представления кода, сгенерированного по Chisel-описанию; сейчас язык развивается в рамках проекта CIRCT. FIRRTL предоставляет конструкции для унифицированного (независимого от HDL) описания цифровой аппаратуры на уровне регистровых передач: схема (circuit), модуль (module), вход (input), выход (output), тактовый сигнал (clock), провод (wire), регистр (reg), блок памяти (mem), условие (when), присваивание (<=), экземпляр модуля (inst) и другие (в том числе разные арифметико-логические операции).

Языки HW, Comb и Seq используются в связке для описания RTL-моделей: HW определяет общую структуру системы, а Comb и Seq – комбинационные и последовательностные схемы,

лежащие в основе модулей. Основными сущностями HW являются модули разных видов (hw.module, hw.module.extern, hw.module.generated) и их экземпляры (hw.instance); Comb определяет арифметико-логические операции (comb.add, comb.and, comb.concat и другие); Seq включает конструкции для работы с регистрами, очередями, блоками памяти (seq.firreg, seq.fifo, seq.firmem и другие).

Язык LLHD (Low-Level Hardware Description) – разработка Швейцарской высшей технической школы Цюриха (ETH Zürich) – задумывался как самодостаточное средство представления цифровой аппаратуры на поведенческом (behavioral), структурном (structural) и вентильном (netlist) уровнях. Несмотря на то, что изначально LLHD возник как отдельный проект, сейчас он развивается в рамках инфраструктуры CIRCT. Основными единицами LLHD являются функции (functions), процессы (processes) и сущности (entities): функции определяют вычисления без побочных эффектов (могут использоваться в разных частях описания); процессы описывают поведение системы (изменение состояния и выходов в зависимости от входов); сущности определяют структуру системы в форме иерархии (в них создаются экземпляры процессов и других сущностей). Среди особенностей языка следует отметить поддержку 9-значной логики, используемой в языке VHDL (IEEE 1164 [29]), и Тьюринг-полноту, что, например, необходимо для представления тестовых окружений.

3.3 Анализ существующих разработок

Результаты анализа существующих языков представлены в табл. 2. Как видно, ни один из них не удовлетворяет полностью сформулированным требованиям: языки MaxJ, DSLX и Пифагор не являются ЯПП (при этом они ориентированы на разработку ПВ и могут служить идейной основой для разработки ЯПП); языки HIR, FIRRTL, HW/Comb/Seq и LLHD не привязаны к потоковой обработке, однако, являясь MLIR-диалектами, они допускают интеграцию с ЯПП других уровней; из них выделяется язык HIR, который основан на более высоком уровне абстракции и имеет механизмы конкретизации временны́х характеристик (фиксирования результатов планирования вычислений).

Табл. 2. Анализ языков, используемых в проектировании потоковых вычислителей.
Table 2. Analysis of the languages used in the design of streaming computers.

Язык / **Требование**	**MaxJ, DSLX, Пифагор**	**HIR**	**FIRRTL, HW/Comb/Seq, LLHD**
1. потоковая обработка данных	+	−	−
2. временная абстракция	+	+	−
3. средства планирования	−	+	−
4. интеграция другими уровнями	−	+	+

ЯПП ПВ разумно строить путем сочетания сильных сторон языков разных типов. На наш взгляд, наиболее перспективная комбинация включает MaxJ (проработанный понятийный аппарат, большой промышленный опыт использования), HIR (механизм временны́х переменных) и диалекты CIRCT (трансформации представления и экспорт HDL-описания).

4. Язык промежуточного представления DFCIR

Предлагаемый ЯПП называется DFCIR (DataFlow Computer Intermediate Representation). Реализация DFCIR основана на инфраструктуре MLIR (DFCIR является MLIR-диалектом). Основные сущности языка заимствованы из MaxJ (см. раздел 3.1): поток, скаляр, счетчик, цепь счетчиков, смещение в потоке и другие.

Применение DFCIR в автоматизации проектирования ПВ описано ниже:

1. высокоуровневое описание ПВ на предметно-ориентированном языке (DSL, Domain-Specific Language) транслируется в DFCIR;
2. на уровне DFCIR осуществляется планирование вычислений и оптимизация под заданные целевые характеристики;
3. оптимизированное DFCIR-представление транслируется в FIRRTL, HW/Comb/Seq или иные ЯПП уровня RTL;
4. RTL-представление экспортируется в HDL (Verilog, SystemVerilog, VHDL).

4.1 Типы данных и операции

Типы данных языка показаны в табл. 3: базовые типы включают числа с фиксированной точкой (`fixed`), числа с плавающей точкой (`float`) и битовые векторы (`rawbits`); составные типы – комплексные числа (`complex`), структуры (`struct`) и векторы (`vector`).

Табл. 3. Типы данных DFCIR.
Table 3. DFCIR data types.

Тип данных	**Синтаксис**	**Описание**

Число с фиксированной точкой	**`fixed`**
	`  $sign,`
	`  $intBits,`
	`  $fracBits>`
		`$sign` – знаковость числа (*true* или *false*);
		`$intBits` – число бит в целой части;
		`$fracBits` – число бит в дробной части.

Число с плавающей точкой	**`float`**
	`  $expBits,`
	`  $fracBits>`
		`$expBits` – число бит в порядке;
		`$fracBits` – число бит в мантиссе.

Битовый вектор	**`rawbits`**`<$width>`	`$width` – длина битового вектора.
Комплексное число	**`complex`**`<$type>`	`$type` – тип вещественной и мнимой частей.
Структура	**`struct`** `  $name`$_1$`:$type`$_1$ `  [, $name`$_i$`:$type`$_i$`]*` >	`$name`$_i$ – имя *i*-ого поля; `$type`$_i$ – тип *i*-ого поля.

Вектор	**`vector`**
	`  $type`
	`  $size>`
		`$type` – тип элементов вектора;
		`$size` – размер вектора.

Над данными определены следующие арифметико-логические операции: получение противоположного значения (`neg`), сложение (`add`), вычитание (`sub`), умножение (`mul`), деление (`div`), остаток от деления (`rem`, `mod`), сравнение на равенство/неравенство (`eq`, `notEq`), сравнение на меньше/больше (`less`, `greater`), сравнение на меньше/больше или равно (`lessEq`, `greaterEq`), сдвиг влево/вправо (`shl`, `shr`), побитовые НЕ, И, ИЛИ, исключающее ИЛИ (`not`, `and`, `or`, `xor`), извлечение битов (`bits`), конкатенация битовых векторов (`cat`), приведение типа (`cast`).

4.2 Основные конструкции

Основные конструкции DFCIR представлены в табл. 4: имена параметров начинаются с символа `$`; запись `$arg*` (`*` в конце является сокращением от `$arg-name : $arg-type`); скобки ***[ ]*** выделяют необязательную часть; ***[ ]**** – повторение нуля или более раз.

Табл. 4. Конструкции DFCIR.
Table 4. DFCIR constructs.

Синтаксис	Описание
Объявления	
kernel $name { $body }	Ядро с именем $name и телом $body.
scalarInput<$type>($name)	Входной скаляр / поток типа $type с именем $name. В случае потока можно задать управляющий (стробирующий) поток $control.
input<$type>($name ***[***, $control****]***)	
scalarOutput<$type>($name) ***[***<= $stream****]***	Выходной скаляр / поток типа $type с именем $name и присоединяемым потоком $stream. В случае потока можно задать управляющий (стробирующий) поток $control.
output<$type>($name ***[***, $control****]*** ***[***<= $stream****]***	
constant<$type> $value	Константа типа $type со значением $value.
Арифметико-логические операции	
UNARY_OP($arg*) : $type	Унарная / бинарная операция над аргументом $arg / аргументами $lhs и $rhs (потоками или скалярами) с указанием типа результата $type.
BIN_OP($lhs*, $rhs*) : $type	
bits($arg*, $lhs, $rhs) : **rawbits**<$width>	Извлечение бит из диапазона [$lhs, $rhs] из скаляра / потока $arg и представление результата в битовом векторе длины $width.
cat($lhs*, $rhs*) : **rawbits**<$width>	Конкатенация битовых представлений $lhs и $rhs и представление результата в битовом векторе длины $width.
cast<$type>($arg*)	Приведение скаляра / потока $arg к типу $type.
Управляющие конструкции	
mux($selector*, $element$_1$ ***[***, $element$_i$****]*****) : $type	Мультиплексор с потоком-селектором $selector и входными потоками $element$_1$, … типа $type. Если $element$_i$ является константой или скаляром, для них должен быть указан их исходный тип.
offset($stream*, $offset*) : $type	Смещение $offset в потоке $stream.
simpleCounter<$type>($max*)	Счетчик со значениями типа $type с верхней границей $max.
createCc	Создание цепи счетчиков.
addCounter($chain, $step, $max*) : $type	Создание и добавление счетчика со значениями типа $type с шагом $step и верхней границей $max в цепь $chain.
Операция соединения	
connect($connecting*, $connectee*)	Соединение скаляра / потока $connecting со скаляром / потоком $connectee.
Операция задержки	
latency [$delay] $stream*	Задержка потока $stream на $delay тактов (вставляется при планировании вычислений).

4.3 Примеры описаний

Описание ядра на языке DFCIR начинается с объявления входов (потоков и скаляров), а заканчивается объявлением выходов. Каждый выход требуется соединить с потоком: это можно сделать прямо в операции **`output`** / **`outputScalar`** или с помощью отдельной операции **`connect`**. Для формирования выходных потоков могут использоваться операции, описанные выше. С потоками данных могут быть ассоциированы управляющие потоки (`control`): значение 1 говорит о готовности данных, 0 – о неготовности.

В листинге 1 приведен пример ядра, вычисляющего значения многочлена $x^2 + 2x$.

```
// Объявление псевдонимов используемых типов.
!ui32t = !dfcir.fixed<false, 32, 0>
!boolt = !dfcir.fixed<false, 1, 0>

// Объявление ядра, вычисляющего значение многочлена второго порядка.
dfcir.kernel "Polynomial2" {
  // Объявление входных потоков.
  %x       = dfcir.input<!ui32t>("x")
  %control = dfcir.input<!boolt>("control")

  // Вычисление значения выражения x*x + x.
  %square  = dfcir.mul(%x: !dfcir.stream<!ui32t>,
                       %x: !dfcir.stream<!ui32t>)
                         : !dfcir.stream<!ui32t>
  %midResult  = dfcir.add(%square: !dfcir.stream<!ui32t>,
                          %x     : !dfcir.stream<!ui32t>)
                                 : !dfcir.stream<!ui32t>

  %result = dfcir.add(%midResult: !dfcir.stream<!ui32t>,
                      %x        : !dfcir.stream<!ui32t>)
                                : !dfcir.stream<!ui32t>

  // Объявление выходного потока.
  %out = dfcir.output<!ui32t>("out", %control: !dfcir.stream<!boolt>)
  dfcir.connect(%out: !dfcir.stream<!ui32t>, %result:
!dfcir.stream<!ui32t>)
}
```

Листинг 1. Ядро, вычисляющее $x^2 + 2x$.
Listing 1. A kernel computing $x^2 + 2x$.

Ядро `Polynomial2` имеет два входных потока, `x` и `control`: по первому передаются 32-битные целые числа, по второму – булевы значения. Для значений потока `x` вычисляются выражения $x^2 + 2x$, которые формируют поток `result`. Выход ядра – поток `out` – соединяется с потоком `result` (данные) и потоком `control` (управление).

В листинге 2 приведен пример описания ядра, вычисляющего скользящее среднее.

Ядро `MovingAverage` имеет входной поток `x`, по которому передаются числа с плавающей точкой, и входной скаляр `size`, задающий число элементов в потоке `x`. Для каждых трех подряд идущих элементов рассчитывается среднее значение (для первого и последнего элементов потока среднее считается по двум значениям – элементы за пределами потока полагаются равными 0). В примере используются конструкция **`offset`** для обращения к элементам до и после текущего. Последний элемент в потоке определяется с помощью счетчика (**`simpleCounter`**). Условия реализованы на мультиплексорах (**`mux`**): например, если текущий элемент является граничным, используется делитель 2, иначе – 3.

```
// Объявление псевдонимов используемых типов.
!ui32t = !dfcir.fixed<false, 32, 0>
!ui32s = !dfcir.stream<!ui32t>
!ui32v = !dfcir.scalar<!ui32t>
!ui32c = !dfcir.const<!ui32t>
!boolt = !dfcir.fixed<false, 1, 0>
!bools = !dfcir.stream<!boolt>
!f32t  = !dfcir.float<8, 23>
!f32s  = !dfcir.stream<!f32t>
!f32c  = !dfcir.const<!f32t>

// Объявление ядра, вычисляющего скользящее среднее.
dfcir.kernel "MovingAverage" {
  // Объявление используемых констант и входных данных.
  %C0   = dfcir.constant<!ui32t> 0: i64
  %C0F  = dfcir.constant<!f32t> 0.0: f32
  %C1   = dfcir.constant<!ui32t> 1: i64
  %C2   = dfcir.constant<!ui32t> 2: i64
  %C3   = dfcir.constant<!ui32t> 3: i64
  %x    = dfcir.input<!f32t>("x")
  %size = dfcir.scalarInput<!ui32t>("size")

  %prevOrig = dfcir.offset(%x, -1: i64) : !f32s
  %nextOrig = dfcir.offset(%x,  1: i64) : !f32s

  // Объявление счетчика и вычисление предикатов.
  %cnt = dfcir.simpleCounter<!ui32t>(%size: !ui32v)
  %abvLowBnd = dfcir.greater(%cnt: !ui32s, %C0: !ui32c) : !bools
  %lessThanSize = dfcir.sub(%size: !ui32v, %C1: !ui32c) : !ui32s
  %blwUppBnd = dfcir.less(%cnt: !ui32s, %lessThanSize: !ui32s) : !bools
  %inBounds = dfcir.and(%abvLowBnd: !bools, %blwUppBnd: !bools) : !bools
  %prev = dfcir.mux(%abvLowBnd: !bools, %C0F: !f32c, %prevOrig) : !f32s
  %next = dfcir.mux(%blwUppBnd: !bools, %C0F: !f32c, %nextOrig) : !f32s

  // Вычисление суммы для подсчета скользящего среднего.
  %divisor = dfcir.mux(%inBounds: !bools, %C2: !ui32c, %C3: !ui32c) : !ui32s
  %sum1 = dfcir.add(%prev: !f32s, %x: !f32s) : !f32s
  %sum2 = dfcir.add(%sum1: !f32s, %next: !f32s) : !f32s
  %result = dfcir.div(%sum2: !f32s, %divisor: !ui32s) : !f32s

  // Объявление выходного потока.
  %y = dfcir.output<!f32t>("y") <= %result: !f32s
}
```

Листинг 2. Ядро, вычисляющее скользящее среднее.
Listing 2. A kernel computing a moving average.

4.4 Трансляция в FIRRTL

В табл. 4 представлено отображение типов DFCIR в типы FIRRTL.

Табл. 4. Трансляция типов DFCIR в типы FIRRTL.
Table 4. DFCIR-to-FIRRTL type translation.

DFCIR	**FIRRTL**
!**dfcir.float**<*m*, *n*>	!**firrtl.uint**<*m* + *n* + 1>
!**dfcir.fixed**<false, *m*, *n*>	!**firrtl.uint**<*m* + *n*>
!**dfcir.fixed**<true, *m*, *n*>	!**firrtl.int**<*m* + *n* + 1>
!**dfcir.rawbits**<*n*>	!**firrtl.uint**<*n*>
!**dfcir.complex**<*type*>	!**firrtl.bundle**<re: *type*, im: *type*>
!**dfcir.struct**<*name_i: type_i, ...*>	!**firrtl.bundle** <*name_i: type_i, ...*>
!**dfcir.vector**<*type*, *size*>	!**firrtl.vector**<*type*, *size*>

В табл. 5 описывается обобщенная схема трансляции конструкций DFCIR в конструкции FIRRTL.

Табл. 5. Трансляция конструкций DFCIR в конструкции FIRRTL.
Table 5. DFCIR-to-FIRRTL construct translation.

DFCIR	**FIRRTL**
dfcir.kernel "*Name*"	**firrtl.module** @*Name*(..., !**firrtl.clock**)
Ядро транслируется в одноименный модуль. Для каждого объявления входного / выходного скаляра / потока создается соответствующий порт в модуле. Интерфейс модуля дополняется портом тактового сигнала (!**firrtl.clock**). Создается одноименная схема (**firrtl.circuit**), в которую помещается созданный модуль.	
dfcir.constant *:*	**firrtl.constant** *:*
dfcir.op(...):(...)	**firrtl.instance**(..., !**firrtl.clock**):(...)
Для всех арифметико-логических операций (за исключением побитовых) объявляются внешние модули (**firrtl.extmodule**) и создаются их экземпляры (**firrtl.instance**); интерфейсы модулей расширены портами тактового сигнала. Побитовые операции DFCIR имеют прямые аналоги в FIRRTL. Выходы одних экземпляров соединяются со входами других с помощью операции **firrtl.connect**.	
dfcir.connect(...)	**firrtl.connect**(...)
dfcir.bits(...)	**firrtl.bits**(...)
dfcir.cat(...)	**firrtl.cat**(...)
dfcir.latency [*d*] ...	**firrtl.instance**(..., !**firrtl.clock**):(...)
Для всех задержек объявляются внешние модули (**firrtl.extmodule**) и создаются их экземпляры (**firrtl.instance**); интерфейсы модулей расширены портами тактового сигнала. Выходы одних экземпляров соединяются со входами других с помощью операции **firrtl.connect**.	

5. Экспериментальный анализ

Реализация DFCIR была внедрена в открытый инструмент синтеза Utopia HLS [30]. Для высокоуровневого описания ПВ в Utopia HLS используется язык DFCxx (оба языка разработаны в рамках одного проекта; рассмотрение DFCxx выходит за рамки статьи). Для экспериментального анализа использовался пример обратного дискретного косинусного преобразования (ОДКП) и методика сравнения, описанная в [12]. По DFCxx-описанию ОДКП 8x8 было построено промежуточное представление на DFCIR; на базе промежуточного представления было выполнено планирование вычислений, после чего было сгенерировано SystemVerilog-описание; далее с помощью инструмента Vivado 2017.4 синтезирована схема для ПЛИС Xilinx Virtex Ultrascale+ (XCVU9P-FLGB2104-2-E). Результаты, дополненные данными из [12], представлены в табл. 6.

Вычислитель синтезированный Utopia HLS, оказался близким по производительности к вычислителю, синтезированному XLS (31.6 и 31.3 млн операций в секунду соответственно), при этом его описание на языке DFCxx на 65 строк меньше. Лидером является инструмент MaxCompiler, однако в отличие от других сравниваемых инструментов он строит схему с интерфейсом PCIe, а не AXI-Stream.

6. Заключение

В работе предложен язык DFCIR, ориентированный на промежуточное представление специализированных ПВ. Система понятий DFCIR заимствована из языка MaxJ, реализация выполнена на основе инфраструктуры MLIR, синтаксис спроектирован с оглядкой на языки

проекта CIRCT, прежде всего FIRRTL. Реализация DFCIR внедрена в инструмент синтеза Utopia HLS [30]; проведено экспериментальное сравнение разработанных средств с другими инструментами (MaxCompiler, XLS, Vivado HLS, Bambu). Результаты показали потенциал предложенного подхода (инструмент уступил лишь MaxCompiler). Среди направлений дальнейшей работы следует выделить следующие: (1) реализация оптимизирующих преобразований моделей ПВ; (2) разработка методов имитационной и формальной верификации ПВ; (3) создание предметно-ориентированных библиотек функциональных элементов для конструирования ПВ.

Табл. 6. Результаты экспериментального анализа инструмента Utopia HLS.
Table 6. Utopia HLS experimental analysis results.

Язык / инструмент	**Лицензия**	**Специализация**	**Число строк в описании**	**Итоговая частота (МГц)**	**Пропускная способность (млн оп/с)**
Verilog	–	Аппаратура общего назначения	316	113.21	14.15
MaxJ / MaxCompiler	Проприетарная	Потоковые вычислители	163	403.13	44.79
DSLX / XLS	Apache License v2.0	Потоковые вычислители	243	250.50	31.31
C / Vivado HLS	Проприетарная	Аппаратура общего назначения	130	131.46	16.43
C / Bambu	GPL v3.0	Аппаратура общего назначения	191	257.33	1.39
DFCxx / Utopia HLS	Apache License v2.0	Потоковые вычислители	178	252.52	31.57

Список литературы / References

[1]. I. Ekmecic, I. Tartalja, V. Milutinovic. EM/sup 3/: a taxonomy of heterogeneous computing systems. Computer, vol. 28, no. 12, 1995. pp. 68-70. DOI: 10.1109/2.476202.

[2]. I. Ekmecic, I. Tartalja, V. Milutinovic. A survey of heterogeneous computing: concepts and systems. Proceedings of the IEEE, vol. 84, no. 8, 1996. pp. 1127-1144. DOI: 10.1109/5.533958.

[3]. IEEE Standard for Verilog Hardware Description Language. IEEE Std 1364-2005, 2006. DOI: 10.1109/IEEESTD.2006.99495.

[4]. IEEE Standard for SystemVerilog: Unified Hardware Design, Specification and Verification Language. IEEE Std 1800-2005, 2005. DOI: 10.1109/IEEESTD.2005.97972.

[5]. IEEE Standard VHDL Language Reference Manual. IEEE Std 1076-2019, 2019. DOI: 10.1109/IEEESTD.2019.8938196.

[6]. С.Ю. Кун. Матричные процессоры на СБИС. Мир, 1991. 672 с.

[7]. E. Lee, D. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on Computers, C-36(1), 1987. pp. 24-35. DOI: 10.1109/TC.1987.5009446.

[8]. C. Leiserson, J. Saxe. Retiming synchronous circuitry. Algorithmica, vol. 6, 1991. pp. 5-35. DOI: 10.1007/BF01759032.

[9]. MLIR: Multi-level Intermediate Representation. URL: https://mlir.llvm.org/ (дата обращения: 15.08.2024).

[10]. The LLVM Compiler Infrastructure Project. URL: https://llvm.org/ (дата обращения: 15.08.2024).

[11]. CIRCT: Circuit IR Compilers and Tools. URL: https://circt.llvm.org/ (дата обращения: 15.08.2024).

[12]. A. Kamkin, M. Chupilko, M. Lebedev, S. Smolov, G. Gaydadjiev. High-level synthesis versus hardware construction. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2023. DOI: 10.23919/DATE56975.2023.10136904.

[13]. Chisel/FIRRTL Hardware Compiler Framework. URL: https://www.chisel-lang.org/

(дата обращения: 15.08.2024).
[14]. MyHDL. URL: https://www.myhdl.org/ (дата обращения: 15.08.2024).
[15]. Bluespec SystemVerilog Documentation. URL: https://web.ece.ucsb.edu/its/bluespec/ (дата обращения: 15.08.2024).
[16]. P. Coussy, M. Meredith, D. Gajski, A. Takach. An Introduction to High-level Synthesis. IEEE Design and Test of Computers, 2009. pp. 8-17. DOI: 10.1109/MDT.2009.69.
[17]. Vitis HLS. URL: https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0090-vitis-hls-hub.html/ (дата обращения: 15.08.2024).
[18]. Bambu: A Free Framework for the High-Level Synthesis of Complex Applications. URL: https://panda.deib.polimi.it/?page_id=31/ (дата обращения: 15.08.2024).
[19]. LegUp: Software IDE for Programming Microchip FPGAs. URL: http://lightsail.legupcomputing.com/ (дата обращения: 15.08.2024).
[20]. MaxCompiler. URL: https://www.maxeler.com/products/software/maxcompiler/ (дата обращения: 15.08.2024).
[21]. XLS: Accelerated HW Synthesis. URL: https://google.github.io/xls/ (дата обращения: 15.08.2024).
[22]. О.В. Непомнящий, А.И. Легалов, И.Н. Рыженко. Метод архитектурно-независимого высокоуровневого синтеза СБИС. Известия ЮФУ. Технические науки, №8 (202), 2018. с. 38-47. DOI: 10.23683/2311-3103-2018-8-38-47.
[23]. M. Kingshuk, B. Uday. HIR: An MLIR-based Intermediate Representation for Hardware Accelerator Description. ACM International Conference on Architectural Support for Programming Languages and Operating Systems (APLOS), vol. 4, 2023. pp. 189-201. DOI: 10.1145/3623278.3624767.
[24]. FIRRTL dialect. URL: https://circt.llvm.org/docs/Dialects/FIRRTL/(дата обращения: 15.08.2024).
[25]. HW dialect. URL: https://circt.llvm.org/docs/Dialects/HW/ (дата обращения: 15.08.2024).
[26]. Comb dialect. URL: https://circt.llvm.org/docs/Dialects/Comb/ (дата обращения: 15.08.2024).
[27]. Seq dialect. URL: https://circt.llvm.org/docs/Dialects/Seq/, (дата обращения: 15.08.2024).
[28]. LLHD dialect. URL: https://circt.llvm.org/docs/Dialects/LLHD/ (дата обращения: 15.08.2024).
[29]. IEEE Standard Multivalue Logic System for VHDL Model Interoperability. IEEE Std 1164-1993, 1993. DOI: 10.1109/IEEESTD.1993.115571.
[30]. Utopia: A High-Level Synthesis Framework. URL: https://github.com/ispras/utopia-hls/ (дата обращения: 15.08.2024).

Информация об авторах / Information about authors

Александр Сергеевич КАМКИН – кандидат физико-математических наук, ведущий научный сотрудник ИСП РАН и РЭУ им. Г.В. Плеханова; преподает в МГУ, МФТИ, НИУ ВШЭ и РЭУ. Научные интересы: формальные методы, синтез и верификация цифровой аппаратуры, гетерогенные компьютерные системы.

Alexander Sergeevich KAMKIN – Cand. Sci. (Phys.-Math.), leading researcher at ISP RAS and Plekhanov Russian University of Economics; teaches at MSU, MIPT, HSE, and PRUE. Research interests: formal methods, synthesis and verification of digital hardware, and heterogeneous computer systems.

Михаил Юрьевич ЛИТВИНОВ – лаборант Отдела технологий программирования ИСП РАН. Научные интересы: автоматизация проектирования гетерогенных компьютерных систем.

Mikhail Yurievich LITVINOV is a laboratory assistant at Software Engineering Department of ISP RAS. Research interests: automated design of heterogeneous computer systems.

Иван Александрович ГРИГОРОВ – инженер Отдела технологий программирования ИСП РАН. Научные интересы: высокоуровневый синтез, модели потоков данных.

Ivan Aleksandrovich GRIGOROV is an engineer assistant at Software Engineering Department of ISP RAS. Research interests: high-level synthesis and dataflow models.