DOI: 10.15514/ISPRAS-2022-34(5)-1

Сравнение инструментов высокоуровневого синтеза и конструирования цифровой аппаратуры

[1,2,3,4,5] *А.С. Камкин, ORCID: 0000-0001-6374-8575 <kamkin@ispras.ru>*
[1,2] *М.М. Чупилко, ORCID: 0000-0002-8772-5631 <chupilko@ispras.ru>*
[1,2] *М.С. Лебедев, ORCID: 0000-0002-0207-7672 <lebedev@ispras.ru>*
[1,2] *С.А. Смолов, ORCID: 0000-0003-0173-3081 <smolov@ispras.ru>*
[2,6] *Г. Гайдаджиев, ORCID: 0000-0002-3678-7007 <g.gaydadjiev@rug.nl>*

[1] *Институт системного программирования имени В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Российский экономический университет имени Г.В. Плеханова, 117997, Россия, г. Москва, Стремянный пер., д. 36*
[3] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*
[4] *Московский физико-технический институт, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*
[5] *НИУ "Высшая школа экономики", 101000, Россия, г. Москва, ул. Мясницкая, д. 20*
[6] *Гронингенский университет Нидерланды, г. Гронинген, 9700 АБ*

Аннотация. Предметно-ориентированные системы, использующие ускорители на базе программируемых логических интегральных схем (ПЛИС), все чаще проектируются при помощи инструментов высокоуровневого синтеза и конструирования аппаратуры. В настоящее время разработчикам доступно множество таких инструментов, как открытых, так и коммерческих. В данной работе проведено экспериментальное сравнение нескольких существующих решений (языков и инструментов), а именно: Verilog (базовое решение), Chisel, Bluespec SystemVerilog (Bluespec Compiler), DSLX (XLS), MaxJ (MaxCompiler) и C (Bambu и Vivado HLS). Сравнение и анализ производились на примере алгоритма обратного дискретного косинусного преобразования матрицы 8×8 (ОДКП), широко используемого, в том числе, в JPEG- и MPEG-декодерах. В качестве метрик сравнения реализаций алгоритма ОДКП использовались: 1) степень автоматизации (насколько меньше исходного кода требуется для описания алгоритма по сравнению с Verilog); 2) контролируемость (возможность достижения заданных характеристик аппаратуры, а именно отношения производительности к занимаемой площади); 3) гибкость (легкость внесения изменений в реализацию с целью достижения определенных характеристик). Для оценки характеристик синтезированных схем в реальном окружении были разработаны AXI-Stream-адаптеры. В исследовании показано, как отдельные оптимизации (настройки инструментов и модификации исходного кода) влияют на производительность системы и занимаемую площадь. Продемонстрировано, насколько важно уметь управлять балансом между пропускной способностью интерфейса и производительностью вычислительного ядра; даны рекомендации по разработке новых инструментов проектирования систем, использующих ускорители на основе ПЛИС.

Ключевые слова: автоматизация проектирования микроэлектроники; предметно-ориентированные вычисления; конструирование аппаратуры; высокоуровневый синтез; программируемые логические интегральные схемы; обратное дискретное косинусное преобразование.

Для цитирования: Камкин А.С., Чупилко М.М., Лебедев М.С., Смолов С.А., Гайдаджиев Г. Сравнение инструментов высокоуровневого синтеза и конструирования цифровой аппаратуры. Труды ИСП РАН, том 34, вып. 5, 2022 г., стр. 7-22. DOI: 10.15514/ISPRAS-2022-34(5)-1

Comparison of High-Level Synthesis and Hardware Construction Tools

[1,2,3,4,5] *A.S. Kamkin, ORCID: 0000-0001-6374-8575 <kamkin@ispras.ru>*
[1,2] *M.M. Chupilko, ORCID: 0000-0002-8772-5631 <chupilko@ispras.ru>*
[1,2] *M.S. Lebedev, ORCID: 0000-0002-0207-7672 <lebedev@ispras.ru>*
[1,2] *S.A. Smolov, ORCID: 0000-0003-0173-3081 <smolov@ispras.ru>*
[2,6] *G. Gaydadjiev, ORCID: 0000-0002-3678-7007 <g.gaydadjiev@rug.nl>*

[1] *Ivannikov Institute for System Programming of the RAS, 25 Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Plekhanov Russian University of Economics, 36 Stremyanny lane, Moscow, 117997, Russia*
[3] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[4] *Moscow Institute of Physics and Technology 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*
[5] *National Research University Higher School of Economics 11 Myasnitskaya Ulitsa, Moscow, 101000, Russia*
[6] *University of Groningen, 9700 AB, Groningen, The Netherlands*

Abstract. Application-specific systems with FPGA accelerators are often designed using high-level synthesis or hardware construction tools. Nowadays, there are many frameworks available, both open-source and commercial. In this work, we attempt to fairly compare several existing solutions (languages and tools), including Verilog (our baseline), Chisel, Bluespec SystemVerilog (Bluespec Compiler), DSLX (XLS), MaxJ (MaxCompiler), and C (Bambu and Vivado HLS). Our analysis has been carried out using a representative example of 8×8 inverse discrete cosine transform (IDCT), a widely used algorithm engaged in, among others, JPEG and MPEG decoders. The metrics under consideration include: (a) the degree of automation (how much less code is required compared to Verilog), (b) the controllability (possibility to achieve given design characteristics, namely a given ratio of the performance and area), and (c) the flexibility (ease of design modification to achieve certain characteristics). Rather than focusing on computational kernels only, we have developed AXI-Stream wrappers for the synthesized implementations, which allows adequately evaluating characteristics of the designs when they are used as parts of real computer systems. Our study shows clear examples of what impact specific optimizations (tool settings and source code modifications) have on the overall system performance and area. It emphasizes how important is to be able to control the balance between the communication interface utilization and the computational kernel performance and delivers clear guidelines for the next generation tools for designing FPGA accelerator based systems.

Keywords: electronic design automation; application-specific computing; hardware construction; high-level synthesis; field-programmable gate array; inverse discrete cosine transform

For citation: Kamkin A.S., Chupilko M.M., Lebedev M.S., Smolov S.A., Gaydadjiev G. Comparison of High-Level Synthesis and Hardware Construction Tools. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 5, 2022. pp. 7-22 (in Russian). DOI: 10.15514/ISPRAS-2022-34(5)-1

1. Введение

Одним из начальных этапов проектирования цифровой аппаратуры является создание модели целевого устройства на уровне регистровых передач (register transfer level, RTL), для чего используются языки описания аппаратуры (hardware description language, HDL), такие как Verilog или VHDL [1]. Эти языки появились в 1980-90-е годы и произвели революцию в

области разработки микроэлектроники. Сейчас (несмотря на некоторое обновление в 2000-х годах) они выглядят устаревшими; например, в них нет высокоуровневых примитивов проектирования, полезных при реализации аппаратных вычислителей в некоторых предметных областях, в том числе в цифровой обработке сигналов (digital signal processing, DSP).

В настоящее время предложены два подхода, нацеленных на повышение продуктивности проектирования микроэлектронных устройств: *высокоуровневый синтез* (high-level synthesis, HLS) и *конструирование аппаратуры* (hardware construction, HC) [2, 3]. В первом случае RTL-модель автоматизированным образом синтезируется из высокоуровневого описания алгоритма. Во втором случае используется более детализированное описание, где микроархитектура устройства задается явно. В обоих случаях разработчик может проводить исследование проектных альтернатив (design space exploration, DSE) и оптимизировать модель в соответствии с заданными ограничениями на производительность, энергопотребление и занимаемую площадь.

Целями данной работы являются:

1) исследование возможностей инструментов HLS/HC для ПЛИС;
2) испытание инструментов на выбранном тестовом примере;
3) оценка эффективности полученных RTL-моделей.

В качестве тестового примера был использован алгоритм обратного дискретного косинусного преобразования (ОДКП) матрицы 8×8 [4]. Пример ОДКП был выбран по следующим соображениям: 1) это известный алгоритм, используемый в декодерах JPEG и MPEG [5, 6]; 2) это «типичная» функция одного вычислительного ядра; 3) существуют готовые реализации: на C [6], DSLX [7] и Bluespec SystemVerilog (BSV) [8]. Алгоритм ОДКП был реализован на нескольких языках, являющихся входными для инструментов HLS/HC (разработанные реализации размещены в открытом репозитории [9]). Полученные реализации (синтезированные схемы) сравнивались по ряду метрик, включая:

1) *степень автоматизации* (насколько меньше исходного кода требуется по сравнению с Verilog);
2) *управляемость* (возможность достижения заданных характеристик аппаратуры);
3) *гибкость* (легкость внесения изменений в модель и/или нахождения параметров инструментов с целью достижения заданных характеристик).

Статья организована следующим образом. В разд. 2 представлена классификация методов HLS/HC и обзор существующих HLS/HC-решений. В разд. 3 описывается методология оценки инструментов (метрики сравнения, тестовый пример и процедура оценки). В разд. 4 представлены результаты экспериментального анализа инструментов. В разд. 5 подведены итоги и описаны направления дальнейших исследований.

2. Обзор работ

Основным подходом к проектированию цифровых СБИС в настоящее время является разработка RTL-моделей на языках Verilog и VHDL (с их последующим логическим и физическим синтезом). В последнее время появляются новые языки и инструменты проектирования. Этому способствует очевидное желание снизить затраты на разработку и верификацию аппаратуры. В зависимости от способа решения этой задачи можно выделить следующие группы инструментов (в скобках приведены примеры):

1) специализированные генераторы, предназначенные для построения эффективных аппаратных реализаций определенных алгоритмов (FloPoCo [10] для арифметических ядер);

2) инструменты конструирования аппаратуры, построенные на основе языков программирования общего назначения, таких как Scala, Python и Java (Chisel [11], MyHDL [12], JHDL [13]);
3) инструменты конструирования аппаратуры на основе специализированных языков (немного более высокого уровня, чем традиционные HDL) (Bluespec Compiler [14], Esterel Studio [15], DIL Compiler [16]);
4) инструменты высокоуровневого синтеза на основе императивных языков программирования, таких как C/C++ и Fortran (Bambu [17], LegUp [18], Vivado HLS [19]);
5) инструменты высокоуровневого синтеза на основе языков параллельного программирования, таких как CUDA C, OpenCL и DPC++ (Vitis HLS [20], Intel FPGA SDK for OpenCL [21], OneAPI [22]);
6) инструменты высокоуровневого синтеза на основе языков программирования потоков данных, основанных на императивной и функциональной парадигмах (MaxCompiler [23], XLS [7]);
7) инструменты высокоуровневого синтеза на основе предметно-ориентированных языков и библиотек, таких как MATLAB, TensorFlow и OpenVX (HDL Coder [24], Vitis AI [25], HiFlipVX [26]).

В последние несколько лет было опубликовано несколько обзоров инструментов HLS/HC [27, 28, 29]. В работе [27] рассмотрены следующие инструменты: AccelDSP, Agility, AutoPilot, Bluespec Compiler (BSC), Catapult C, Compaan, C-to-Silicon, CyberWorkBench, DK Design Suite, Impulse CoDeveloper, ROCCC и Synphony C. Сравнение инструментов проводилось по следующим характеристикам:

1) язык описания и уровень абстракции;
2) легкость разработки и процесса обучения;
3) поддержка чисел с плавающей и фиксированной точкой;
4) легкость исследования проектных альтернатив (опции оптимизации, модификация исходного кода и т.д.);
5) возможности для верификации (наличие/отсутствие встроенных генераторов тестовых окружений);
6) использованные для синтезированных схем ресурсы ПЛИС.

В качестве примера использовался фильтр Собеля 3×3. На данный момент этот обзор несколько устарел (появились новые инструменты). Кроме того, обзор посвящен в основном оценке продуктивности разработки; в нем отсутствует информация, актуальная для нашего исследования – насколько хорошо инструменты могут оптимизировать по сравнению с RTL-моделями, разработанными вручную.

В работе [28] авторы концентрируются на использовании инструментов HLS/HC для разработки гетерогенных компьютерных систем, использующих ускорители на основе ПЛИС. Существующие инструменты были разделены на 4 категории в зависимости от языка описания:

1) C/C++ и их подмножества: Trident, GAUT, Streams-C, Impulse-C, FpgaC, NAPA C, Nimble, CHiMPS, CASH, LegUp, ROCCC, C2H, ASC, Catapult C, AutoPilot, DEFACTO, Carte, DIME-C, Bash-C, Mitrion-C, SpecC, SPC и SPARK;
2) не C/C++-подобные языки: MATLAB (MATCH), Python (MyHDL), Java (JHDL, Sea Cucumber), C# (Kiwi), Pebble, Esterel и BSV;
3) языки визуального моделирования: LabVIEW, Simulink и Altium Designer;
4) языки, ориентированные на графические процессоры: CUDA (FCUDA) и OpenCL (SOpenCL).

В обзоре отсутствует сравнение инструментов HLS/HC друг с другом. Многие упомянутые в нем средства либо уже не поддерживаются, либо не доступны для скачивания.

В работе [29] авторы провели подробный анализ актуальных инструментов HLS/HC и оценили некоторые из них (компиляторы из C в Verilog/VHDL). В обзоре рассмотрены инструменты eXCite, Cynthesizer, CHC, MaxCompiler, Bambu, DWARV, Vivado HLS, gcc2verilog, HercuLeS, Garp, PipeRench, SA-C, CtoVerilog, а также средства, перечисленные в статьях [27, 28]. Сравнение проводилось по следующим характеристикам:

1) тип лицензии;
2) входной и выходной языки;
3) область применения;
4) автоматизация тестирования;
5) поддержка чисел с фиксированной и плавающей точкой.

Отличительная особенность работы – описание базовых оптимизаций HLS/HC. В статье представлены результаты экспериментального сравнения трех академических инструментов (Bambu, DWARV и LegUp) и одного коммерческого (не назван). Авторы провели два набора экспериментов на выбранных тестовых примерах: сначала инструменты синтеза запускались с настройками по умолчанию; затем примеры модифицировались с целью максимизации производительности и инструменты синтеза запускались на измененных примерах с применением опций оптимизации. В обоих случаях для синтезированных RTL-моделей оценивались тактовая частота, общая задержка и количество использованных LUT, DSP-блоков и BRAM. Большинство тестовых примеров было взято из набора CHStone [30].

В заключение обзора отметим, что существующие обзоры либо устарели, либо содержат только общие сведения о возможностях инструментов HLS/HC, либо описывают эксперименты для небольшого набора инструментов. Данная работа призвана устранить эти недостатки.

3. Методология сравнения

В данном разделе описывается методология оценки инструментов HLS/HC.

3.1 Метрики

Для сравнения используются следующие первичные показатели:

- *размер исходного кода* (L) – количество строк кода (lines of code, LOC), включая настройки соответствующих инструментов HLS/HC (директивы, параметры, опции и т.д.);
- *производительность*, или *пропускная способность* (P), – количество операций в секунду (operations per second, OPS);
- *площадь*, или *объем использованных ресурсов* (A), – количество задействованных LUT и триггеров (flip-flop, FF) на ПЛИС.

Количество строк кода можно считать объективной метрикой, хотя и не идеальной. По всей видимости, она может быть адаптирована к языкам визуального программирования (т.к. инструменты HLS/HC должны генерировать код на каком-то этапе), хотя они не рассматриваются в данной работе.

Отношение $Q = \frac{P}{A}$ будем называть *качеством* модели, или *эффективностью.* Пусть Ф – целевая функция (далее будем считать, что $\Phi \equiv Q$).

Пусть $\mathcal{A}$ – алгоритм (тестовый пример), $\mathcal{L}$ – язык описания аппаратуры и $\mathcal{T}$ – инструмент HLS/HC на основе $\mathcal{L}$.

1) *Степень автоматизации*: α показывает, насколько легко (в терминах количества строк кода) описать $\mathcal{A}$ с помощью $\mathcal{L}$ и настроить $\mathcal{T}$ в сравнении с ручной разработкой на Verilog (L_V):

$$\alpha = \frac{(L_V - L)}{L_V} \times 100\%. \quad (1)$$

2) *Управляемость*: набор средств, позволяющих разработчику аппаратуры влиять на результаты синтеза (в терминах Ф) без изменения функциональности. Включает в себя настройки инструмента, аннотации кода и сам исходный код. Метрика управляемости C_Φ определяется по следующей формуле:

$$C_\Phi = \frac{\Phi^*}{\Phi_V^*} \times 100\%, \quad (2)$$

где Φ^* – максимальное значение Ф, полученное инструментом $\mathcal{T}$, а Φ^*_V – «абсолютный» максимум (ожидаемый при использовании Verilog-модели, разработанной вручную).

3) *Гибкость*: F_Φ показывает, насколько легко улучшить значение Ф путем настройки $\mathcal{T}$ и изменения исходного кода:

$$F_\Phi = \frac{\Phi^* - \Phi_0}{\Delta L}, \quad (3)$$

где значения Φ^* и Φ_0 соответствуют «оптимальному» и «начальному» вариантам модели, а $\Delta L = \Delta L^+ + \Delta L^-$ – это количество измененных строк кода (добавленных или удаленных), включая аннотации и опции инструмента.

3.2 Тестовый пример

В качестве тестового примера используется ОДКП матрицы 8×8. Преобразование определяется следующей формулой:

$$f_{ij} = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C_u C_v F_{uv} \cos\left(\frac{\pi}{8}\left(i+\frac{1}{2}\right)u\right)\cos\left(\frac{\pi}{8}\left(j+\frac{1}{2}\right)v\right), \quad (4)$$

где $F \in \mathbb{R}^{8\times8}$ – входное изображение в частотной области, $f \in \mathbb{R}^{8\times8}$ – выходное изображение в пространственной области и

$$C_k = \begin{cases} 1/\sqrt{2}, & k = 0, \\ 1, & k > 0. \end{cases}$$

Формулу (4) можно представить в виде 16 одномерных преобразований: 8 по строкам и 8 по столбцам матрицы. Каждое из них определяется формулой:

$$g_i = \frac{1}{2}\sum_{u=0}^{7} C_u G_u \cos\left(\frac{\pi}{8}\left(i+\frac{1}{2}\right)u\right), \; i = \overline{0,7}. \quad (5)$$

Алгоритм ОДКП 8×8 представлен ниже:

```
for i = 0,7 do temp_i* ← IDCT^row(F_i*) end // по строкам
for i = 0,7 do f_ij ← IDCT^col(temp_*j) end // по столбцам
```

Алгоритм 1. ОДКП 8×8
Algorithm 1. IDCT 8×8

Здесь x_{k*} (x_{*k}) означает k-ю строку (столбец) матрицы x. На вход алгоритму подается матрица 12-битных чисел; на выходе формируется матрица 9-битных чисел:

$F_{uv} \in$ [-2,048, 2,047], $f_{ij} \in$ [-256, 255].

Рассматриваемые реализации следуют представленной выше схеме, но вместо применения формулы (5) для IDCT^{row} и IDCT^{col} они используют алгоритм Чена-Вана с использованием «бабочки» (11 умножений, 29 сложений и несколько сдвигов) [31, 32]. Все реализации совместимы со стандартом IEEE 1180-1990 [4] и разрабатывались на основе реализации ОДКП из набора тестов на соответствие ISO/IEC 13818-4:2004 [6, 33].

Заметим также, что IDCT^{row} и IDCT^{col} используют разные коэффициенты: $C_k^{row} = C_k \times 256\sqrt{2}$ и $C_k^{col} = \frac{C_k}{256\sqrt{2}}$. Все реализации, разработанные в рамках данного исследования, размещены в открытом репозитории [9].

3.3 Процедура сравнения

Для всех рассмотренных в данном исследовании нотаций (Verilog, Chisel, BSV, DSLX, MaxJ, C) «начальная» модель была разработана в соответствии с алгоритмом [6]. Для получившегося функционального блока вычислялось количество строк исходного кода (без учета комментариев и пустых строк) – L^{FU}. «Оптимальная» модель (для целевой функции Ф) разрабатывалась путем применения опций и/или прагм инструментов и модификации исходного кода. Для оптимизации моделей применялась конвейеризация вычислений.

Для того, чтобы получить AXI-Stream-совместимые [34] модели (что распространено для библиотек IP-блоков [35]), были разработаны адаптеры интерфейса. Входные и выходные матрицы передаются построчно через вход `TDATA` (с помощью протокола с подтверждением установления связи через `TVALID/TREADY`). Адаптеры создавались либо автоматически инструментом HLS/HC, либо вручную на языке разработки или Verilog. Для второго случая подсчитывалось количество строк кода адаптера – L^{AXI}. Единственное отступление от данной процедуры было сделано для языка MaxJ и инструмента MaxCompiler. Последний работает только на системном уровне и генерирует PCIe-адаптер. В этом случае рассматривалось вычислительное ядро без адаптера интерфейса ($L^{AXI} = 0$).

Если инструмент требовал использования дополнительных настроек, то учитывались размер конфигурационных файлов и количество параметров – L^{Conf}. Для экспериментов с начальными версиями описаний применялись настройки по умолчанию ($L^{Conf} = 0$).

Трудозатраты на создание функционального блока вычислялись как $L = L^{FU} + L^{AXI} + L^{Conf}$. Степень автоматизации (α) вычислялась по формуле (1).

Для синтеза на ПЛИС использовался коммерческий инструмент Vivado Design Suite 2017.4 [19] с настройками по умолчанию. Для оценки производительности измерялось минимальное значение тактового сигнала (T_{clk}), для которого синтез завершался успешно, и вычислялись максимальная частота и пропускная способность:

$$v_{max} = \frac{1}{(T_{clk} - T_{wns})}, \quad P = \frac{v_{max}}{T_P},$$

где T_{wns} – *максимальное время запаздывания* (worst negative slack, WNS), вычисленное Vivado, и T_P – *периодичность*, т.е. минимальное количество тактов между запусками двух последовательных операций.

В экспериментах также измерялась *задержка* (T_L), т.е. количество тактов исполнения одной операции (включая передачу входных/выходных данных).

Для оценки площади, занимаемой синтезированной схемой на ПЛИС, брались следующие показатели:

- N_{LUT} – количество LUT;
- N_{FF} – количество триггеров;
- N_{DSP} – количество DSP-блоков;
- N_{BRAM} – количество блоков памяти BRAM;
- N_{IO} – количество входных/выходных пинов.

Так как при физическом синтезе различные модели по-разному отображаются на ресурсы ПЛИС (особенно на DSP-блоки), использовался нормализованный показатель:

$$A = N_{LUT}^* + N_{FF}^*,$$

где N_{LUT}^* и N_{FF}^* – количество задействованных LUT и триггеров соответственно при запрете использования DSP-блоков инструментом синтеза (опция `maxdsp=0` в Vivado).

Для полученных значений P и A вычислялось значение целевой функции Ф. Для оценки возможностей инструментов HLS/HC модели оптимизировались с целью максимизации Ф, после чего определялись показатели $C_Ф$ и $F_Ф$ по формулам (2) и (3).

Табл. 1. Исследуемые языки и инструменты
Table 1. Languages and tools under evaluation

Язык	Парадигма	Инструмент	Тип	Лицензия
Verilog	Классический RTL	Vivado	LS/PR	Коммерческая
Chisel	Функциональный/RTL	Chisel	HC	Открытая
BSV	Основанный на правилах/RTL	BSC	HC	Открытая
DSLX	Функциональный	XLS	HLS	Открытая
MaxJ	Потоковый	MaxCompiler	HLS	Коммерческая
C	Императивный	Bambu	HLS	Открытая
		Vivado HLS	HLS	Коммерческая

4. Экспериментальный анализ

Рассмотренные языки и инструменты представлены в табл. 1 (где LS/PR – logic synthesis/place & route, т.е. *логический синтез/размещение и трассировка*). Во всех экспериментах для синтеза схем, оценки их характеристик и генерации двоичных образов для ПЛИС был использован инструмент Vivado Design Suite. Синтез проводился для ПЛИС Xilinx Virtex Ultrascale+ (XCVU9P-FLGB2104-2-E), имеющей следующие характеристики: $N_{LUT} = 1182240$, $N_{LUTRAM} = 591840$, $N_{FF} = 2364480$, $N_{DSP} = 6840$, $N_{BRAM} = 2160$, $N_{IO} = 702$.

Результаты анализа сгруппированы по входным языкам (см. ниже). Количественные данные представлены на рис. 1 и в табл. 2.

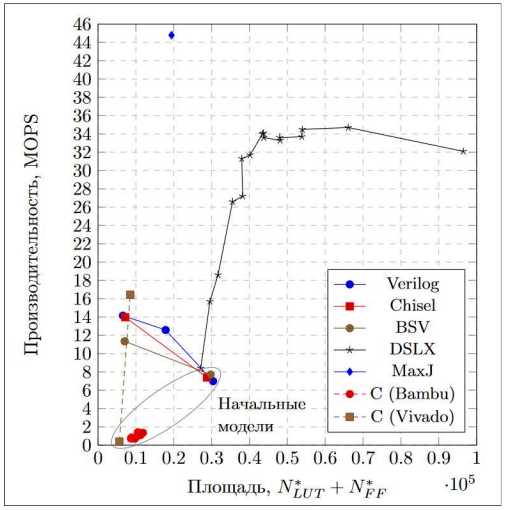

Рис. 1. Исследование проектных альтернатив для ОДКП
Fig. 1. Design space exploration for IDCT

4.1 Verilog

Verilog – язык описания аппаратуры, созданный в середине 1980-х годов и предназначенный для проектирования цифровых СБИС на уровне регистровых передач [36]. Модель представляет собой иерархию модулей; каждый из которых имеет входные и выходные порты и содержит объявления переменных, параллельные процессы (блоки `always` и непрерывные присваивания) и экземпляры других модулей. Для переменных требуется явно указывать их разрядность. Блоки описываются в С-подобном синтаксисе. Язык поддерживает параметризацию модулей и конструкцию `generate` для порождения экземпляров модулей.

Начальная модель. Сначала была разработана простая комбинационная схема с построчным накоплением входных данных через AXI-Stream-адаптер. Схема обрабатывает матрицу целиком и содержит восемь экземпляров модуля $IDCT^{row}$ и восемь экземпляров модуля $IDCT^{col}$, что приводит к большой занимаемой площади (30 396) и низкой частоте (55,88 МГц). Самым узким местом в этой реализации является последовательный адаптер (теоретически, пропускная способность схемы в 8 раз выше).

Оптимизации. Были разработаны две оптимизированные модели. Первая состоит из одного экземпляра $IDCT^{row}$ и восьми $IDCT^{col}$ (поскольку за один такт работы схемы на вход поступает только одна строка данных, то достаточно одного экземпляра $IDCT^{row}$). Пропускная способность выросла в 1,8 раза (за счет повышения частоты), а площадь уменьшилась в 1,7 раза. Соответственно, качество выросло более чем в 3 раза.

Вторая реализация состоит из одного экземпляра $IDCT^{row}$ и одного $IDCT^{col}$. Задержка схемы выросла с 17 до 24 тактов, в то время как пропускная способность увеличилась в 2 раза (по сравнению с начальной моделью), площадь уменьшилась в 4,6 раза, а качество выросло в 9,4 раза.

4.2 Chisel

Chisel (Constructing Hardware in a Scala Embedded Language) [11] – язык описания аппаратуры на уровне регистровых передач, основанный на Scala. В добавление к обычным конструкциям языка описания аппаратуры, он предоставляет возможности объектно-ориентированного и функционального программирования (полиморфизм, абстрактные типы данных, рекурсия), что позволяет разрабатывать на нем гибкие генераторы компонентов СБИС [3]. Язык поддерживает вывод типов: разрядность портов и регистров задается вручную, а размеры остальных переменных могут быть вычислены автоматически. Следует заметить, что Chisel работает в связке с промежуточным представлением FIRRTL (Flex Intermediate Representation for RTL) [37].

Начальная модель. Как и в случае с Verilog, сначала была реализована простая комбинационная схема. Данная реализация имеет несколько большую производительность по сравнению с Verilog (105,7%) и при этом меньшую площадь (94,6%). Это можно объяснить тем, что в Verilog-описании используется 32-битная арифметика (как в [6]), в то время как Chisel автоматически (и точнее) определяет разрядность переменных.

Оптимизации. Была разработана конвейерная реализация с одним модулем $IDCT^{row}$ и одним модулем $IDCT^{col}$. Она оказалась немного хуже аналогичной Verilog-реализации (производительность – 98,7%, площадь – 109,5%). Следует отметить, что модель на Chisel была разработана в 2-3 раза быстрее.

4.3 Bluespec SystemVerilog (BSC)

Bluespec SystemVerilog (BSV) – основанный на правилах язык описания аппаратуры, разработанный в начале 2000-х [14]. BSV-описания являются иерархическими, но модули описываются не так, как на Chisel или Verilog. В каждом модуле определены элементы состояния и правила, меняющие это состояние. Правило состоит из условия срабатывания и

атомарного действия, описывающего изменение состояния. BSV находится на более высоком уровне абстракции по сравнению с Verilog и Chisel (в терминах описания времени): модель программирования (операционная семантика) подразумевает выполнение одного правила за такт, однако компилятор может оптимизировать модель и запланировать выполнение нескольких правил на один такт.

Были разработаны две модели на BSV. По ним были синтезированы 26 реализаций путем варьирования опций инструмента BSC и применения атрибутов в коде. Оказалось, что настройки оказывают небольшое влияние (менее 1%) на производительность и площадь, поэтому данные ниже соответствуют настройкам инструмента по умолчанию.

Начальная модель. Начальная BSV-реализация была разработана на основе С-реализации [6]. Полученная модель оказалась несколько лучше начальных реализаций на Verilog и Chisel (производительность – 110,3%, площадь – 97,2% по сравнению с Verilog).

Оптимизации. Оптимизированная версия также основана на конвейеризации. Ее качество оказалось немного ниже: производительность – 80,2%, а площадь – 107,1% относительно Verilog. Главная причина меньшей производительности – в том, что показатель периодичности на один такт больше (9 вместо 8). Теоретически данная особенность может быть устранена, что позволит получить характеристики, сравнимые с характеристиками Chisel-реализации.

4.4 DSLX (XLS)

DSLX – потоковый функциональный язык для описания вычислительных ядер [38]. Он похож на язык Rust, но учитывает аппаратные особенности (объекты фиксированного размера, полностью анализируемые графы вызовов). Поддерживаемые типы данных: битовые векторы, кортежи, структуры и массивы. Можно отметить другие детали: параметрические структуры и функции, вывод типов, конструкции `for` (циклы с фиксированным числом итераций). Будучи высокоуровневым языком, DSLX игнорирует временны́е свойства, позволяя компилятору планировать вычисления (разбивать граф вычислений по стадиям конвейера).

Для экспериментов был адаптирован существующий пример ОДКП [7] (были изменены разрядности элементов входной и выходной матриц). Были синтезированы 19 реализаций путем применения следующих опций: 1) тип схемы (комбинационная или конвейерная) и 2) количество стадий конвейера.

Начальная модель: Сначала была создана комбинационная схема с AXI-Stream-адаптером, разработанным вручную. Эта реализация заметно лучше, чем все начальные реализации, представленные ранее: производительность – 120,3%, а площадь – 89,2% относительно Verilog.

Оптимизации: Эксперименты показали, что максимальное качество достигается, когда количество стадий конвейера равно 8 (по неизвестным причинам исполнение операции при этом занимает 3 такта): производительность – 221,2%, а площадь – 578,1% по сравнению с конвейерной Verilog-реализацией. Показатель качества в 38,3% выглядит не очень хорошо; вероятно, проблема в последовательном адаптере интерфейса (пропускная способность могла бы быть в 8 раз выше).

4.5 MaxJ (MaxCompiler)

MaxJ – потоковый императивный язык программирования для разработки высокопроизводительных систем [39]. Системы, проектируемые на этом языке, состоят из трех частей: 1) вычислительные ядра, 2) менеджер (соединяющий ядра друг с другом, ЦПУ и памятью) и 3) управляющее программное обеспечение. Ядра и менеджер описываются на MaxJ. Язык основан на Java и предназначен для описания графов потоков данных (dataflow graphs), в которых могут быть узлы следующих типов: значения (константы и параметры

времени исполнения), вычислительные узлы (арифметические и логические операции), смещения (доступ к “прошлым” и “будущим” элементам потоков данных), мультиплексоры (узлы принятия решений), счетчики (организация циклов) и входы/выходы.

В отличие от других рассмотренных инструментов, MaxCompiler позволяет разработать систему целиком, а не только аппаратную часть на ПЛИС; предоставляет программный интерфейс и библиотеку времени исполнения для передачи данных на вычислительные ядра и управления вычислениями. Программная и аппаратная части соединены через шину PCIe. Таким образом, не совсем корректно сравнивать MaxJ-реализации с другими реализациями.

Начальная модель. Сначала была разработана модель, которая на каждом такте принимает и выдает матрицу 8×8 целиком. Синтезированная реализация содержит 47 конвейерных стадий и может работать на частоте 403,13 МГц, самой высокой среди всех реализаций. Здесь узким местом является шина PCIe. Пропускная способность оценивается как пропускная способность шины PCIe 3.0 x 16 (около 16 ГБ/с), поделенная на размер входных данных (1024 бита). Несмотря на то, что реализация требует много ресурсов, ее качество (963%) существенно выше, чем у начальной Verilog-реализации.

Оптимизации. Для снижения занимаемой площади было разработано другое вычислительное ядро, которое за такт получает на вход одну строку матрицы и сохраняет промежуточные результаты в памяти ПЛИС. В отличие от начального варианта, в этом случае производительность ограничена частотой. Занимаемая площадь сократилась в 2,8 раза, пропускная способность – в 2,7 раза, а качество повысилось на 4%.

4.6 C (Bambu и Vivado HLS)

C [40] – императивный язык программирования, разработанный в начале 1970-х годов. Язык широко применяется в различных областях, включая системное программирование, вычислительную математику и цифровую обработку сигналов. Базовыми элементами программ на языке C являются объявления данных и функции. Язык поддерживает целые числа и числа с плавающей точкой, а также указатели, массивы, структуры и объединения; есть условные операторы и циклы. Так как C предполагает один поток исполнения, по C-программам сложно синтезировать высокопроизводительные устройства.

Были проведены эксперименты с двумя компиляторами из C в Verilog: открытым Bambu и коммерческим Vivado HLS. В обоих случаях C-код [6] был модифицирован: округление в $IDCT^{col}$ было реализовано в виде функции (`iclip`), а не массива фиксированных значений [9]. Bambu, в отличие от Vivado HLS, не умеет генерировать AXI-Stream-адаптеры, поэтому адаптер был разработан вручную на Verilog.

4.6.1 Bambu

Инструмент Bambu предоставляет широкий набор опций, покрывающий все этапы высокоуровневого синтеза: планирование операций, выделение ресурсов и т.д. В данном исследовании применялся набор настроек `experimental-setup`, предназначенный для синтеза схем, оптимальных по разным критериям (производительность, площадь и баланс между ними). Было установлено, что большинство опций не оказывает ощутимого влияния на качество синтезируемых схем. Также был обнаружен странный эффект – хранение данных в BRAM (поведение по умолчанию) приводит к некорректному функционированию модели (при отключении этой опции модель работает корректно). Всего были исследованы 42 конфигурации.

Начальная модель. Сначала была использована следующая конфигурация: `channels-type=MEM_ACC_11` (один канал на чтение и один канал на запись в память) и `memory-allocation-policy=LSS` (локальные/статические переменные и строки хранятся в BRAM). Несмотря на относительно высокую частоту (471,4% относительно начальной

Verilog-реализации), схема, будучи последовательной, имеет низкую производительность (11,7%) при небольшой площади (29,2%).

Оптимизации. Наилучшее качество было достигнуто на конфигурации `BAMBU-PERFORMANCE-MP` (два канала на чтение и два канала на запись в память) в сочетании с опциями `speculative-sdc-scheduling` и `memory-allocation-policy=LSS`. Полученные характеристики близки к показателям начальной модели и существенно хуже характеристик оптимизированных реализаций, построенных по моделям на других языках: производительность – 9,8%, площадь – 160,1% по сравнению с Verilog.

4.6.2 Vivado HLS

В инструменте Vivado HLS процесс синтеза управляется директивами препроцессора (прагмами). Существуют директивы конвейеризации, встраивания функций, развертки циклов, оптимизации массивов, упаковки структур, синтеза интерфейса и т.д. Отметим, что инструмент может автоматически генерировать требуемый интерфейс. Для создания AXI-Stream-адаптера была добавлена отдельная функция и прагма `#pragma HLS INTERFACE axis port=⟨name⟩`.

Начальная модель. Реализация, синтезированная с настройками по умолчанию, имеет не слишком высокие характеристики: пропускная способность почти в 18 раз меньше, чем у начальной Verilog-реализации. Основная проблема состоит в том, что инструмент не подставляет код функций $IDCT^{row}$ и $IDCT^{col}$ в места их вызова и генерирует лишние AXI-Stream-интерфейсы между ними.

Оптимизации. Для преодоления описанной выше проблемы исходный код был модифицирован: массив `short buf[8]` в $IDCT^{row}$ и $IDCT^{col}$ был заменен на явно заданную последовательность элементов `short buf0, … short buf7`. Также была добавлена прагма `HLS PIPELINE`. В результате, качество синтезированной реализации вплотную приблизилось к оптимизированной Verilog-реализации (89,7%): производительность – 116,1%, площадь – 129,5%.

Табл. 2. Результаты оценки инструментов HLS/HC
Table 2. HLS/HC tools evaluation results

Язык	**Verilog**		**Chisel**		**BSV**		**DSLX**	
Инструмент	**Vivado**		**Chisel**		**BSC**		**XLS**	
Конфигурация	*Нач.*	*Опт.*	*Нач.*	*Опт.*	*Нач.*	*Опт.*	*Нач.*	*Опт.*
LOC, с опциями	247	316	195	222	238	199	242	243
Модификации, ΔL	258		131		434		**3**	
Автоматизация, α	0%	0%	21,1%	29,8%	3,6%	37,0%	2,0%	23,1%
Качество, $Q = P/A$	230	2155	257	1942	259	1614	310	825
Контролируемость, C_Q	100%		90.1%		74.8%		38.3%	
Гибкость, F_Q	7.5		12.9		3.1		**171.7**	
Частота, МГц	55,88	113,21	59,15	111,77	100,25	102,18	67,30	250,50
Производительность, MOPS	6,99	14,15	7,39	13,97	7,71	11,35	8,41	31,31
Задержка, тактов	**17**	24	**17**	24	21	26	**17**	19
Периодичность, тактов	8	8	8	8	13	9	8	8
Площадь, $N_{LUT^*}+N_{FF^*}$	30396	6567	28778	7194	29549	7036	27127	37965
N_{LUT^*} (maxdsp=0)	29059	**3909**	27441	4530	27565	4781	25805	26960
N_{FF^*} (maxdsp=0)	1337	2658	1337	2664	2184	2255	**1322**	11005
N_{LUT}	13850	**2106**	9283	2205	26560	4643	25805	26010
N_{FF}	1337	2658	1337	2661	2184	2255	**1322**	10717
N_{DSP}	160	20	184	23	40	4	0	16
N_{IO}	172	170	172	172	174	172	170	170

Язык	**MaxJ**		**C**			
Инструмент	**MaxCompiler**		**Bambu**		**Vivado HLS**	
Конфигурация	*Нач.*	*Опт.*	*Нач.*	*Опт.*	*Нач.*	*Опт.*
LOC, с опциями	**121**	163	183	191	125	130
Модификации, Δ*L*	231		29		71	
Автоматизация, *α*	51,4%	48,4%	25,9%	39,6%	49,0%	**58,9%**
Качество, *Q = P/A*	2215	**2308**	91	132	69	1933
Контролируемость, C_Q	**100% [107,1%]**		6,1%		89,7%	
Гибкость, F_Q	0,4		1,4		26,3	
Частота, МГц	**403,13**	**403,13**	263,44	257,33	132,61	131,46
Производительность, MOPS	**123,08**	44,79	0,82	1,39	0,39	16,43
Задержка, тактов	47	60	323	185	340	26
Периодичность, тактов	1	9	323	185	340	8
Площадь, $N_{LUT^*}+N_{FF^*}$	55580	19413	8879	10514	**5633**	8501
N_{LUT^*} (maxdsp=0)	19704	5941	5443	7134	4103	4974
N_{FF^*} (maxdsp=0)	35876	13472	3436	3380	1530	3527
N_{LUT}	19704	5941	5090	6614	2334	3123
N_{FF}	35876	13472	3436	3156	1481	3346
N_{DSP}	384	48	5	9	22	19
N_{IO}	59	59	174	174	270	267

4.7 Резюме

Данные, полученные в результате экспериментов, представлены на рис. 1 и в табл. 2. Рисунок показывает результаты исследования проектных альтернатив в пространстве *Производительность×Площадь*. В таблице содержится информация о начальных и оптимизированных реализациях, полученных с помощью разных инструментов HLS/HC. Жирным цветом выделены наилучшие значения характеристик.

Максимальный уровень *автоматизации* достигается инструментами MaxCompiler и Vivado HLS. Первый – это специализированный инструмент; второй можно считать наилучшим решением для быстрого прототипирования. Говоря об *управляемости* (и *качестве* полученных реализаций), можно отметить инструменты Chisel и BSC; Vivado HLS тоже показывает хороший результат. Самые *гибкие* инструменты – это XLS и Vivado HLS, хотя они конфигурируются по-разному: у первого используется всего один параметр (количество стадий конвейера), у второго – прагмы в исходном коде.

Пример ОДКП, в силу своей простоты, использует немного ресурсов ПЛИС; таким образом, полученные результаты не могут быть однозначно экстраполированы на более сложные модели. Для сложных схем, требующих практически всех ресурсов ПЛИС, инструменты размещения и трассировки ведут себя по-другому, пытаясь «вписать» схему в ПЛИС с помощью различных оптимизаций. Данная работа посвящена более ранним этапам проектирования.

5. Заключение

В статье рассмотрены инструменты высокоуровневого синтеза и конструирования аппаратуры, использующие разные входные нотации: Chisel (основанный на Scala язык описания на уровне регистровых передач), BSC (язык спецификации аппаратуры на базе правил), XLS (Rust-подобный функциональный язык), MaxCompiler (основанный на Java язык описания потоков данных), Bambu и Vivado HLS (оба поддерживают язык общего назначения C). Для каждого инструмента на примере ОДКП были оценены уровень автоматизации, управляемость и гибкость.

Исследование показало, что самым сбалансированным решением (среди рассмотренных) является коммерческий инструмент Vivado HLS. Другим перспективным решением может быть использование открытых расширяемых платформ. Концепция такого инструмента может выглядеть следующим образом. На верхнем уровне находятся *предметно-ориентированные языки* и связанные с ними механизмы трансляции и оптимизации. Далее идут *языки общего назначения*, ориентированные на параллельные вычисления (например, MaxJ) и соответствующее промежуточное представление (вычислительный граф). Отдельные блоки могут быть разработаны с помощью *низкоуровневых инструментов*, как универсальных (XLS, Chisel, BSC, Verilog), так и специализированных (FloPoCo). Важной функцией является способность генерировать внешние и внутренние интерфейсы (как в Vivado HLS). Разработка такого инструмента – основная цель нашей дальнейшей работы.

Список литературы / References

[1] Botros N.M. HDL Programming Fundamentals: VHDL and Verilog. Charles River Media, 2005, 506 p.

[2] Coussy P., Gajski D.D. et al. An introduction to high-level synthesis. IEEE Design & Test of Computers, vol. 26, issue 4, 2009, pp. 8-17.

[3] Bachrach J., Vo H. et al. Chisel: constructing hardware in a Scala embedded language. In Proc. of the Design Automation Conference (DAC), 2012, pp. 1216-1225.

[4] IEEE Standard Specifications for the Implementations of 8x8 Inverse Discrete Cosine Transform. In IEEE Std 1180-1990, 1991, pp. 1-12.

[5] Information technology – Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF) – Part 5: ISO/IEC 10918-5:2013, 2013.

[6] MPEG-2 decoder from ISO/IEC 13818-4:2004. Available at: https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_13818-4_2004_Conformance_Testing/Video/verifier/mpeg2decode_960109.tar.gz, accessed 02.11.2022.

[7] XLS. Available at: https://github.com/google/xls, accessed 02.11.2022.

[8] Bluespec Compiler. Available at: https://github.com/B-Lang-org/bsc, accessed 02.11.2022.

[9] IDCT algorithm implementations. Available at: https://github.com/ispras/hls-idct, accessed 02.11.2022.

[10] de Dinechin F., Pasca B. Designing custom arithmetic data paths with FloPoCo. IEEE Design & Test of Computers, vol. 28, issue 4, 2011, pp. 18-27.

[11] Chisel. Available at: https://github.com/chipsalliance/chisel3, accessed 02.11.2022.

[12] MyHDL. Available at: https://www.myhdl.org, accessed 02.11.2022.

[13] Bellows P., Hutchings B. JHDL-an HDL for reconfigurable systems. In Proc. of the IEEE Symposium on FPGAs for Custom Computing Machines, 1998, pp. 175-184.

[14] Bluespec SystemVerilog Reference Guide, Bluespec, Inc., 2017, 421 p.

[15] Berry G., Gonthier G. The ESTEREL synchronous programming language: design, semantics, implementation. Science of Computer Programming, vol. 19, issue 2, 1992, pp. 87-152.

[16] Goldstein S., Budiu M. Fast compilation for pipelined reconfigurable fabrics. In Proc. of the ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays, 1999, pp. 195-205.

[17] Bambu. Available at: https://github.com/ferrandi/PandA-bambu, accessed 02.11.2022.

[18] LegUp. Available at: http://legup.eecg.utoronto.ca, accessed 02.11.2022,

[19] SmartHLS Compiler. Available at: https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/smarthls-compiler, accessed 02.11.2022.

[20] Vivado Design Suite User Guide Implementation. UG904 (v2021.1), August 30, 2021. Available at: https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2021_1/ug904-vivado-implementation.pdf, accessed 02.11.2022.

[21] Vitis High-Level Synthesis User Guide. UG1399 (v2021.2), December 15, 2021. Available at: https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2021_2/ug1399-vitis-hls.pdf, accessed 02.11.2022.

[22] Intel FPGA SDK for OpenCL. Available at: https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html, accessed 02.11.2022.

[23] Intel oneAPI. Available at: https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html, accessed 02.11.2022.

[24] MaxCompiler. Available at: https://www.maxeler.com/products/software/maxcompiler, accessed 02.11.2022.

[25] HDL Coder. Available at: https://www.mathworks.com/products/hdl-coder.html, accessed 02.11.2022.

[26] Vitis AI. Available at: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html, accessed 02.11.2022.

[27] Kalms L., Podlubne A., Göhringer D. HiFlipVX: an Open Source High-Level Synthesis FPGA Library for Image Processing. Lecture Notes in Computer Science, vol. 11444, 2019, pp. 149-164.

[28] Meeus W., Van Beeck K. et al. An overview of today's high-level synthesis tools. Design Automation for Embedded Systems, vol. 16, 2012, pp. 31-51.

[29] Daoud L., Zydek D., Selvaraj H. A survey of high level synthesis languages, tools, and compilers for reconfigurable high performance computing. Advances in Intelligent Systems and Computing, vol. 240, 2014, pp. 483-492.

[30] Nane R., Sima V.-M. et al. A survey and evaluation of FPGA high-level synthesis tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, issue 10, 2016, pp. 1591-1604.

[31] Hara Y., Tomiyama H. et al. CHStone: A benchmark program suite for practical C-based high-level synthesis. In Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS), 2008, pp. 1192–1195.

[32] Chen W.H., Smith C.H., Fralick S.C. A fast computational algorithm for the discrete cosine transform. IEEE Transactions on Communications, vol. 25, issue 9, 1977, pp. 1004-1009.

[33] Wang Z. Fast algorithm for the discrete W transform and for the discrete Fourier transform. IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 32, issue 4, 1984, pp. 803-816.

[34] Information technology – Generic coding of moving pictures and associated audio information – Part 4: Conformance testing, ISO/IEC 13818-4:2004, December 2004.

[35] AMBA 4 AXI4-Stream Protocol Specification. ARM, Cambridge, UK, ARM IHI 0051A (ID030610), March 03, 2010. Available at: https://developer.arm.com/documentation/ihi0051/a/, accessed 02.11.2022.

[36] Vivado Design Suite User Guide: Model-Based DSP. Design Using System Generator. UG897 (v2020.2), November 18, 2020. Available at: https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2020_2/ug897-vivado-sysgen-user.pdf, accessed 02.11.2022.

[37] IEEE Standard for Verilog Hardware Description Language, IEEE Std 1364-2005, April 7, 2006, 509 p.

[38] FIRRTL. Available at: https://github.com/chipsalliance/firrtl, accessed 02.11.2022.

[39] DSLX Reference. Available at: https://google.github.io/xls/dslx_reference, accessed 02.11.2022.

[40] Multiscale Dataflow Programming. Maxeler Technologies, London, UK, Version 2021.1, May 14, 2021.

[41] Information technology – Programming languages – C, ISO/IEC 9899:2018, June 2018.

Информация об авторах / Information about authors

Александр Сергеевич КАМКИН – кандидат физико-математических наук, ведущий научный сотрудник отдела технологий программирования ИСП РАН, ведущий научный сотрудник научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова; преподает в МГУ им. М.В. Ломоносова, МФТИ и НИУ ВШЭ. Область научных интересов: автоматизация проектирования цифровой аппаратуры, верификация и тестирование.

Alexander Sergeevich KAMKIN is a leading researcher at the Software Engineering Department of ISP RAS, leading researcher at the Heterogeneous Computing Systems research lab of Plekhanov RUE. He is also a lecturer at MSU, MIPT, and HSE. His research interests include digital hardware design automation, verification and testing. Alexander has a PhD degree in Physics and Mathematics.

Михаил Михайлович ЧУПИЛКО – кандидат физико-математических наук, старший научный сотрудник отдела технологий программирования ИСП РАН, старший научный сотрудник научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова. Область научных интересов: автоматизация проектирования цифровой аппаратуры, верификация и тестирование.

Mikhail Mikhaylovich CHUPILKO is a senior researcher at the Software Engineering Department of ISP RAS, senior researcher at the Heterogeneous Computing Systems research lab of Plekhanov RUE. His research interests include digital hardware design automation, verification and testing. Mikhail has a PhD degree in Physics and Mathematics.

Михаил Сергеевич ЛЕБЕДЕВ – научный сотрудник ИСП РАН, старший научный сотрудник научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова; преподает в НИУ ВШЭ. Область научных интересов: высокоуровневый синтез, методы исследования проектных альтернатив, методы оптимизации, машинное обучение, цифровая аппаратура, методы верификации цифровой аппаратуры.

Mikhail Sergeevich LEBEDEV is a researcher at ISP RAS and the "Heterogeneous computer systems" laboratory of Plekhanov RUE. Research interests: high-level synthesis, design space exploration methods, neural networks, digital hardware, hardware verification.

Сергей Александрович СМОЛОВ – научный сотрудник отдела технологий программирования ИСП РАН, старший научный сотрудник научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова. Область научных интересов: автоматизация проектирования цифровой аппаратуры, верификация и тестирование.

Sergey Aleksandrovich SMOLOV is a researcher at the Software Engineering Department of Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS), senior researcher at the Heterogeneous Computing Systems research lab of Plekhanov RUE. His research interests include digital hardware design automation, verification and testing.

Георги ГАЙДАДЖИЕВ – профессор кафедры инновационных компьютерных архитектур Гронингенского университета и почетный приглашенный профессор Имперского колледжа Лондона, заведующий научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова. Область научных интересов: архитектура и микроархитектура вычислительных систем, реконфигурируемые и гетерогенные вычисления, распределенные системы памяти, инструменты и методологии проектирования.

Georgi GAYDADJIEV received his M.Sc. (1996) in Electrical Engineering and Ph.D. (2007) in Computer Engineering from Delft University of Technology. He is currently a full professor in Innovative Computer Architectures at the University of Groningen and an honorary visiting professor at Imperial College, head of the "Heterogeneous computer systems" laboratory of Plekhanov RUE. His research interests include: Computer (System) Architecture and Micro-architecture, Reconfigurable and Heterogeneous Computing, Distributed and Advanced Memory Systems, Design tools and Methodologies, Low-Power Architectures, Architectural Support for Compilers and Runtime Systems.