DOI: 10.15514/ISPRAS-2021-33(6)-12

Реализация искусственных нейронных сетей на ПЛИС с помощью открытых инструментов

[1,2] *М.С. Лебедев, ORCID: 0000-0002-0207-7672 <lebedev@ispras.ru>*
[1] *П.Н. Белецкий, ORCID: 0000-0003-2072-958X <belecky@ispras.ru>*

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

Аннотация. Искусственные нейронные сети широко распространены в современном мире. Для их исполнения используются различные устройства: от микропроцессоров до ПЛИС и заказных СБИС. Важной проблемой при этом является ускорение исполнения нейронных сетей. В этой области на данный момент существует множество открытых инструментов. В данной статье содержится обзор нескольких открытых инструментов для исполнения, ускорения нейронных сетей и синтеза аппаратуры по ним. Некоторые из рассмотренных инструментов были выбраны для апробации на ПЛИС. Для этого было разработано пять тестовых моделей нейронных сетей. Процессор Intel, графический процессор NVIDIA и ПЛИС Cyclone V использовались для проведения экспериментов. Результаты показали, что инструменты TVM/VTA и LeFlow оказались способны довести тестовые модели до исполнения на ПЛИС. Однако результаты исполнения показали, что в большинстве случаев ПЛИС проигрывает в быстродействии другим платформам.

Ключевые слова: искусственный интеллект; нейронные сети; специализированные ускорители; высокоуровневый синтез; ПЛИС; открытое программное обеспечение.

Для цитирования: Лебедев М.С., Белецкий П.Н. Реализация искусственных нейронных сетей на ПЛИС с помощью открытых инструментов. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 175-192. DOI: 10.15514/ISPRAS–2021–33(6)–12

Artificial Neural Network Inference on FPGAs Using Open-Source Tools

[1,2] *M.S. Lebedev, ORCID: 0000-0002-0207-7672 <lebedev@ispras.ru>*
[1] *P.N. Belecky, ORCID: 0000-0003-2072-958X <belecky@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *Plekhanov Russian University of Economics, 36, Stremyanny lane, Moscow, 117997, Russia.*

Abstract. Artificial neural networks are widely spread in the modern world. Various hardware is used for neural network inference: from CPUs and GPUs to FPGAs and ASICs. An important research area is inference acceleration. Many open-source tools have been proposed in this area. This article contains a review of a range of open-source tools for neural network inference, acceleration and hardware synthesis. Some of the tools have been selected for evaluation on an FPGA. Five neural network examples have been used as test models. Intel CPU, NVIDIA GPU and Cyclone V FPGA have been used as evaluation platforms. Results show that TVM/VTA and LeFlow tools can successfully process neural network models and run them on the FPGA. However, execution results are controversial.

Keywords: artificial intelligence; neural networks; custom accelerators; high-level synthesis; FPGA; open-source

For citation: Lebedev M.S., Belecky P.N. Artificial Neural Network Inference on FPGAs Using Open-Source Tools. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 175-192 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–12

1. Введение

Искусственный интеллект все больше проникает в повседневную жизнь. Нейронные сети широко используются в обработке мультимедиа, медицине, беспилотном транспорте, общественной безопасности и т.д. В настоящее время уже существует множество открытых (*open-source*) инструментов и библиотек для разработки, хранения, распространения и исполнения нейронных сетей. Наиболее значимые из них – TensorFlow [1], PyTorch [2] и Caffe [3]. А формат ONNX [4] часто применяется для хранения и передачи моделей нейронных сетей и поддерживается большим количеством инструментов.

Важными проблемами в сфере искусственных нейронных сетей являются их оптимизация, исполнение и ускорение на различных аппаратных платформах: мобильных и стационарных микропроцессорах, графических процессорах, тензорных процессорах, ускорителях машинного обучения, а также ПЛИС и проблемно-ориентированных СБИС.

Решением проблемы ускорения может быть использование особых видов нейронных сетей: разреженных [5][6] или бинаризованных [7][8]. Также возможна разработка новых алгоритмов вычислений [9][10] (например, на основе свертки Винограда), техник удаления [11] или слияния [12] слоев нейронной сети, квантования весов [13][14], новых архитектур [15][16] ускорителей и методов оптимизации [17][18] моделей нейронных сетей. Разработчики многих предлагаемых методов заявляют о существенном ускорении вычислений нейронных сетей, в том числе на ПЛИС.

Существует множество открытых инструментов для оптимизации, исполнения и ускорения нейронных сетей, некоторые из которых реализуют описанные выше подходы. Эти инструменты могут быть условно поделены на три группы: 1) инструменты, оптимизирующие модели нейронных сетей для исполнения на устройстве с фиксированной архитектурой; 2) инструменты, оптимизирующие модели нейронных сетей для исполнения на специализированном сопроцессоре (расположенном на ПЛИС); 3) инструменты, позволяющие синтезировать RTL-модель исходной нейронной сети для последующего исполнения на ПЛИС или реализации в виде СБИС.

В этой статье приведен обзор наиболее популярных на сегодняшний день открытых инструментов исполнения и ускорения нейронных сетей, а также результаты экспериментальной апробации нескольких из них на примере моделей двух обученных нейронных сетей (полносвязной и сверточной), а также трех синтетических примеров матричного умножения. Заметим, что процесс обучения нейронных сетей в данной работе не рассматривается.

В разд. 2 данной статьи приведен короткий обзор современных форматов и библиотек для представления моделей нейронных сетей. Разд. 3 посвящен обзору инструментов исполнения и ускорения нейронных сетей. В разд. 4 описаны тестовые модели, с помощью которых проводилась апробация выбранных инструментов. В разд. 5 приведены результаты экспериментов. Разд. 6 – заключение.

2. Форматы представления нейронных сетей

В табл. 1 приведены рассматриваемые в статье форматы и библиотеки представления моделей нейронных сетей.

2.1 ONNX

ONNX (Open Neural Network Exchange) [4] – де-факто общепринятый формат хранения и передачи моделей нейронных сетей. Разрабатывается сообществом ведущих технологических компаний, таких как Intel, AMD, Qualcomm, ARM, Google и Microsoft. Большинство представленных в этой статье инструментов поддерживает входные модели в этом формате. ONNX представляет нейронную сеть в виде графа вычислений, состоящего из входов и выходов, операционных слоев, переменных, типов данных и т.д. Модель нейронной сети сохраняется в двоичном виде. Значения весов и смещений закодированы непосредственно в графе вычислений.

Табл. 1. Форматы представления нейронных сетей
Table 1. Neural Network Frameworks and Formats

Название	Лицензия	Разработчик	Год начала разработки	Формат
ONNX	Apache 2.0	Сообщество компаний	2017	Двоичный
TensorFlow	Apache 2.0	Google	2015	Текстовый / двоичный
Keras	Apache 2.0	Google	2015	Текстовый / двоичный
PyTorch	BSD	Facebook	2016	Текстовый / двоичный
Caffe	BSDv2	Калифорнийский университет в Беркли	2014	Текстовый / двоичный
Caffe2	BSD	Facebook	2017	Текстовый / двоичный
CNTK	MIT	Microsoft	2015	Текстовый / двоичный
CoreML	BSDv3	Apple	2017	Двоичный
MXNet	Apache 2.0	Apache	2016	Текстовый / двоичный
Theano	BSDv3	Монреальский университет	2011	Текстовый / двоичный

2.2 TensorFlow/Keras

TensorFlow [1] – созданная компанией Google платформа разработки приложений машинного обучения. TensorFlow позволяет пользователям описывать, обучать, сохранять и исполнять модели нейронных сетей, а также выполнять тензорные вычисления. Платформа состоит из нескольких библиотек и инструментов, покрывающих различные аспекты разработки приложений машинного обучения. Большинство рассмотренных нами инструментов также поддерживает входные модели в формате TensorFlow. В TensorFlow существует формат хранения моделей SavedModel, позволяющий сохранить полную структуру нейронной сети, включая веса и подграфы.

Модели на TensorFlow описываются по слоям и могут быть оптимизированы и исполнены либо с помощью встроенной среды запуска, либо с помощью компилятора XLA (см. раздел 3). Для исполнения моделей TensorFlow использует *«активное исполнение»* (*eager execution*), т.е. операции в нейронной сети выполняются немедленно, без построения графа вычислений.

Keras [19] – часть платформы TensorFlow, ориентированная на быструю разработку моделей нейронных сетей, их исполнение и портирование на другие платформы. В Keras используется формат H5 для хранения моделей нейронных сетей.

2.3 PyTorch

PyTorch [2] – платформа для тензорных вычислений и разработки моделей нейронных сетей, разработанная компанией Facebook. Приложения, разработанные на PyTorch, в основном ориентированы на исполнение на графических процессорах. Модели нейронных сетей в PyTorch основаны на *обратном автодифференцировании* [20] (*reverse-mode auto-differentiation*), позволяющем динамически менять поведение этих сетей. Модели на PyTorch могут быть сохранены в формате Python *pickle* или формате ONNX.

2.4 Caffe/Caffe2

Caffe [3] – платформа для машинного обучения, разработанная в Калифорнийском университете в Беркли (UC Berkeley). Структура нейронной сети в Caffe описывается слой за слоем в текстовом виде, а не в виде программы. Каждый слой описывает отдельную операцию (от непосредственно слоя нейронной сети или функции активации до входов данных и вспомогательных функций). Данные между слоями передаются с помощью специального представления массивов. Если данные передаются от входа к выходу сети, то осуществляется вычисление результата. Если данные передаются в обратном направлении, то выполняется обучение. Обученные нейронные сети сохраняются в двоичном формате.

Caffe2 [21] – дальнейшее развитие платформы Caffe. Разрабатывается компанией Facebook и входит в состав платформы PyTorch.

2.5 CNTK

CNTK (Microsoft Cognitive Toolkit) [22] – набор инструментов машинного обучения, разрабатываемый компанией Microsoft. Модели нейронных сетей в этом наборе представляются в виде направленных графов вычислений. Листовые вершины этих графов представляют собой входные значения и параметры, внутренние вершины – различные матричные операции.

2.6 CoreML

CoreML [23] – формат представления нейронных сетей в программах и на устройствах компании Apple. С этим форматом предоставляется набор инструментов, с помощью которого можно преобразовывать модели нейронных сетей из сторонних форматов в формат CoreML, а также оптимизировать и запускать их на устройствах Apple.

2.7 MXNet

MXNet [24] – платформа машинного обучения, разрабатываемая сообществом программистов и поддерживаемая фондом Apache. Одной из главных особенностей этой платформы является возможность символического программирования и исполнения нейронных сетей для их оптимизации. Модель на MXNet может быть сериализована в виде двух файлов: структуры нейронной сети и ее весов.

2.8 Theano

Theano [25] – библиотека языка Python для тензорных и матричных вычислений, разработанная в Монреальском университете (Université de Montréal). Ее разработка была прекращена, а результаты использованы в новом проекте по оптимизации тензорных вычислений Aesara [26].

3. Инструменты исполнения и ускорения нейронных сетей

В табл. 2 представлены рассматриваемые в данной статье открытые инструменты исполнения и ускорения нейронных сетей. Их можно условно разделить на три группы.

1) Инструменты, оптимизирующие и исполняющие модели нейронных сетей на целевых устройствах с фиксированными архитектурой и набором инструкций, таких как микропроцессоры, графические процессоры, тензорные процессоры, процессоры цифровой обработки сигналов, микроконтроллеры и т.д. В качестве результата своей работы эти инструменты выдают исполняемый файл под целевую архитектуру, либо некоторое промежуточное представление (например, граф вычислений, веса и представление операций), которое может быть исполнено с помощью среды времени выполнения на целевом устройстве. Инструменты, представленные в этой категории: OpenVINO, PlaidML, MACE, XLA, Glow, TVM.

2) Инструменты, оптимизирующие и исполняющие модели нейронных сетей на специальных сопроцессорах-ускорителях, размещенных на ПЛИС. Обычно эти инструменты предоставляют среду времени выполнения, управляющую передачей входных данных на сопроцессор и считыванием результатов вычислений. В данной категории представлены: TVM с сопроцессором VTA и Vitis AI с семейством ядер DPU.

3) Инструменты, позволяющие синтезировать непосредственно RTL-модель нейронной сети, либо транслирующие ее в формат, пригодный для генерации RTL-модели (например, C/C++ или LLVM) с помощью инструментов высокоуровневого синтеза. Эта категория состоит из: LeFlow (трансляция в LLVM и синтез с помощью открытого инструмента LegUp 4.0 [27]), hls4ml, FINN, ONNC (трансляция в C++ и синтез с помощью коммерческого инструмента Vivado [28]), NNgen (непосредственно синтез Verilog).

Табл. 2. Инструменты исполнения и ускорения нейронных сетей
Table 2. Neural Network Acceleration Tools

Инструмент	Лицензия	Разработчик	Год начала разработки
OpenVINO	Apache 2.0	Intel	2018
PlaidML	Apache 2.0	Intel	2017
MACE	Apache 2.0	Xiaomi	2017
TVM	Apache 2.0	Вашингтонский университет, Apache	2017
XLA	Apache 2.0	Google	2017
LeFlow	BSD	Университет Британской Колумбии	2018
Glow	Apache 2.0	Facebook	2017
hls4ml	Apache 2.0	ЦЕРН и др.	2017
NNgen	Apache 2.0	Шинья Такамаэда-Ямадзаки	2017
ONNC	BSDv3	Skymizer	2018
Vitis AI	Apache 2.0	Xilinx	2020
Finn	BSDv3	Xilinx	2018

3.1 OpenVINO

OpenVINO [29] – набор инструментов для ускорения нейронных сетей, разрабатываемый компанией Intel. В качестве целевых устройств поддерживаются только устройства Intel: процессоры, графические процессоры и процессоры компьютерного зрения (Vision

Processing Units, VPUs). Некоторые версии OpenVINO также поддерживали ПЛИС Intel Arria. На вход OpenVINO может принимать модели нейронных сетей в форматах ONNX, TensorFlow, Caffe, MXNet и Kaldi [30]. Входные модели транслируются в промежуточное представление и оптимизируются: удаляются лишние слои и группируются операции. Далее модель нейронной сети компилируется в исполняемый файл для целевой архитектуры.

3.2 PlaidML

PlaidML [31] – тензорный компилятор, разрабатываемый компанией Intel. Способен компилировать модели нейронных сетей в форматах ONNX, Keras и nGraph [32] для запуска на графических процессорах Intel, AMD и NVIDIA. Для процессоров NVIDIA позволяет не строить промежуточное представление модели на CUDA.

3.3 MACE

MACE (Mobile AI Compute Engine) [33] – платформа для исполнения и ускорения нейронных сетей, разрабатываемая компанией Xiaomi и в основном ориентированная на мобильные устройства (процессоры ARM, графические процессоры Adreno и т.д.). На вход MACE может принимать модели нейронных сетей в форматах ONNX, TensorFlow и Caffe. Встроенный интерпретатор обрабатывает граф вычислений и тензорные операции и передает их среде времени выполнения. Эта среда распределяет вычисления между целевыми устройствами.

3.4 TVM

TVM [34] – компилятор нейронных сетей, разработанный Вашингтонским университетом (University of Washington) и ныне поддерживаемый фондом Apache. TVM позволяет компилировать модели нейронных сетей под различные устройства, такие как микропроцессоры, графические процессоры, микроконтроллеры и т.д. С помощью сопроцессора VTA (Versatile Tensor Accelerator) [35] можно осуществлять исполнение нейронных сетей на ПЛИС. TVM предоставляет средства адаптации компилятора под произвольную целевую архитектуру.

TVM может принимать на вход наибольшее число форматов моделей нейронных сетей из всех рассмотренных в этой статье инструментов: ONNX, TensorFlow, Keras, PyTorch, MXNet, CoreML и другие. Входные модели транслируются во внутреннее представление TVM Relay IR, состоящее из множества *функций* – разновидностей графов вычислений с потоком управления, рекурсией и поддержкой сложных структур данных. После оптимизации функции делятся на *подфункции*, или *сегменты*, состоящие из низкоуровневых операций. Структура исходной модели сохраняется в виде последовательности вызовов подфункций. Каждая подфункция может быть оптимизирована и скомпилирована независимо от других. Наиболее низкоуровневые и архитектурно-зависимые оптимизации проводятся уже компиляторами под целевую архитектуру (например, LLVM или CUDA C). Затем сегменты транслируются в программный код (на C, CUDA, OpenCL, LLVM). Последовательность вызовов преобразуется в формат JSON, а веса сохраняются в двоичном формате. Модель целиком может быть сохранена в виде разделяемой библиотеки и загружена средой времени выполнения TVM, работающей на целевом устройстве и управляющей исполнением модели.

3.4.1 VTA

Сопроцессор VTA – это небольшое тензорное ядро, состоящее из следующих элементов (см. рис. 1):

1) модуль выборки инструкций (Instruction Fetch Module), загружающий инструкции сопроцессора из памяти (DRAM), декодирующий их и распределяющий по очередям;

2) модуль загрузки (Load Module), загружающий входные данные и веса из памяти;

3) модуль сохранения (Store Module), загружающий в память результаты вычислений;

4) вычислительный модуль (Compute Module), осуществляющий перемножение матриц и тензорные операции и, в свою очередь, состоящий из:

   a) регистрового файла (Register File);

   b) кэша микроопераций (Micro-op Cache);

   c) тензорного АЛУ (Tensor ALU);

   d) блока умножения матриц (GEMM Core);

   e) очередей и буферов.

Набор инструкций сопроцессора VTA состоит из операций загрузки и сохранения, умножения матриц и арифметических операций над тензорами.

Внутренние параметры сопроцессора могут быть сконфигурированы: размер элемента входных данных, размер элемента матрицы весов, размеры буферов и т.д.

Существуют две реализации сопроцессора VTA для различных отладочных плат. Реализация на C++ ориентирована на синтез с помощью коммерческого инструмента высокоуровневого синтеза Vivado для плат Xilinx PYNQ (система-на-кристалле Zynq-7000) и Avnet Ultra96 (система-на-кристалле Xilinx Zynq UltraScale+). Реализация на языке Chisel ориентирована на синтез с помощью инструмента Quartus 18.1 для платы Terasic DE10-Nano (ПЛИС Intel Cyclone V).

Для работы с сопроцессором VTA требуется среда времени выполнения, работающая на центральном процессоре платы и осуществляющая получение, обработку и передачу в память DRAM входных данных и получение результатов вычислений из нее.

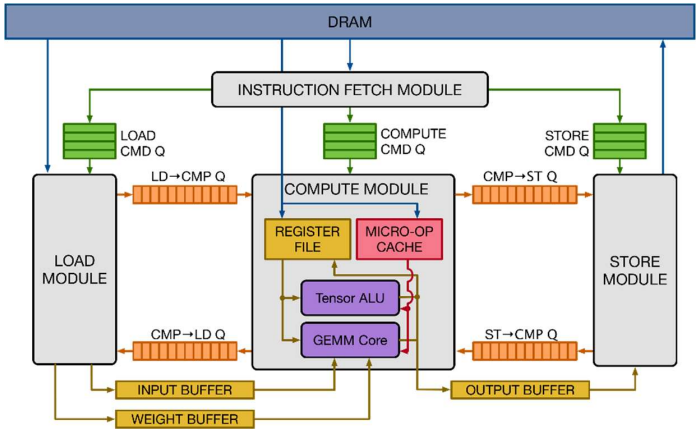

Рис. 1. Структура сопроцессора VTA
Fig. 1. VTA core structure

3.5 XLA

XLA [36] – оптимизирующий компилятор нейронных сетей, входящий в состав платформы TensorFlow. XLA позволяет ускорять исполнение нейронных сетей по сравнению со стандартной средой времени выполнения TensorFlow. Среда времени выполнения TensorFlow представляет операции нейронной сети в виде вычислительных ядер без построения графа вычислений. XLA, напротив, анализирует и оптимизирует граф

вычислений нейронной сети. Он может сливать несколько операций в одно вычислительное ядро и позволяет уменьшить количество обращений в память путем хранения данных в регистрах целевого устройства. XLA осуществляет статическую (ahead-of-time) компиляцию, что позволяет избежать использования среды времени выполнения.

XLA может компилировать модели нейронных сетей в форматах TensorFlow, PyTorch, Julia [37], JAX [38], Nx [39]. Входная модель транслируется во внутреннее представление XLA HLO (High Level Operations), которое затем оптимизируется. Далее HLO-модель трансформируется в LLVM-модель для целевой архитектуры. В качестве целевых архитектур выступают графические процессоры NVIDIA и различные архитектуры микропроцессоров.

3.6 LeFlow

LeFlow [40] – инструмент и маршрут высокоуровневого синтеза RTL-моделей нейронных сетей, разработанный в Университете Британской Колумбии (University of British Columbia). Маршрут (см. рис. 2) позволяет синтезировать Verilog-модель по TensorFlow-модели нейронной сети. Сначала TensorFlow-модель компилируется в неоптимизированное LLVM-представление с помощью XLA. Затем инструмент LeFlow трансформирует и оптимизирует это представление. Наконец, открытый инструмент высокоуровневого синтеза LegUp 4.0 синтезирует RTL-модель нейронной сети. Синтез с помощью LegUp представлен на рис. 2 синими прямоугольниками.

LeFlow трансформирует высокоуровневое представление модели нейронной сети в более приближенное к аппаратуре представление. Добавляются структуры данных для представления тактового сигнала, сигналов сброса, начала работы, завершения работы, выходных данных, создаются глобальные регистры для оптимизации обращений в память и т.д. Так как инструмент LegUp 4.0 не поддерживает некоторые LLVM-операции, сгенерированные XLA, то LeFlow поддерживает несколько измененную версию TensorFlow 1.6 для избегания этих операций. Разработчики LeFlow также добавили несколько собственных оптимизаций в инструмент LegUp 4.0.

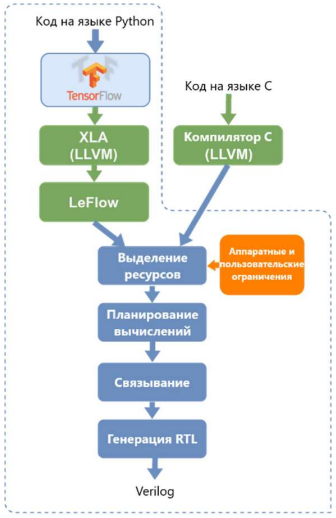

Рис. 2. Маршрут синтеза LeFlow
Fig. 2. LeFlow synthesis route

3.7 Glow

Glow [41] – компилятор нейронных сетей, часть платформы PyTorch. Поддерживает компиляцию под микропроцессоры и графические процессоры. Glow может принимать на вход модели нейронных сетей в форматах ONNX и Caffe2. На основе входной модели строится двухуровневое внутреннее представление. Первое, высокоуровневое, представление – это граф вычислений, сходный с Caffe2-представлением. Операции нейронной сети представляются в виде одной или нескольких вершин графа вычислений. Затем Glow проводит упрощение вершин: операционные вершины трансформируются в вершины низкоуровневых операций линейной алгебры. Далее строится второе, низкоуровневое, представление. На этом уровне операции трансформируются в инструкции. И, наконец, Glow генерирует машинный код. На каждом уровне внутреннего представления Glow проводит различные оптимизации графа вычислений.

3.8 hls4ml

hls4ml [42] – инструмент синтеза C++-модели нейронной сети для инструмента Vivado, разрабатываемый сообществом исследователей из ЦЕРН и нескольких университетов США. hls4ml может обработать модели нейронных сетей в форматах ONNX, Keras, TensorFlow и PyTorch. Инструмент использует библиотеку RFNoC [43] для представления операций нейронной сети. Эта библиотека содержит в себе набор шаблонов нейросетевых операций на C++, оптимизированных для синтеза с помощью прагм инструмента Vivado. hls4ml строит граф вычислений на основе входной модели и вычисляет параметры инстанциации операций. Помимо выходной C++-модели, инструмент генерирует управляющие скрипты для Vivado.

3.9 NNgen

NNgen [44] – инструмент синтеза RTL-моделей из высокоуровневых моделей нейронных сетей, разработанный Шинья Такамаэда-Ямадзаки из Токийского университета (University of Tokyo). NNgen принимает на вход модели в формате ONNX, либо в виде описания на основе собственного предметно-ориентированного языка и библиотеки языка Python.

3.10 ONNC

ONNC [45] – компилятор нейронных сетей под разные целевые устройства (микро- и графические процессоры, процессоры цифровой обработки данных, СБИС и т.д.), разработанный компанией Skymizer. В качестве входного формата нейронных сетей ONNC поддерживает формат ONNX. Инструмент транслирует входную модель в промежуточное представление, оптимизирует ее и разделяет на части, планирует вычисления и выделяет необходимую память. Затем промежуточное представление транслируется в битовый код LLVM для целевой архитектуры. ONNC может транслировать модель нейронной сети в код на языке C, используя собственные реализации операций.

3.11 Vitis AI

Vitis AI [46] – среда для исполнения нейронных сетей на ПЛИС Xilinx, разработанная этой же компанией. Vitis AI состоит из компилятора, оптимизатора, квантователя, профилировщика, среды времени выполнения и набора сопроцессоров глубокого обучения DPU (Deep Learning Processing Unit). Vitis AI не является полностью открытым: для оптимизатора требуется коммерческая лицензия, а сопроцессоры DPU зашифрованы и практически все доступны только в платной версии инструмента Vivado. Целевыми платформами для Vitis AI являются системы-на-кристалле Alveo, Zynq UltraScale+ и ограниченно Zynq-7000. На вход Vitis AI может принимать модели нейронных сетей в форматах TensorFlow, Caffe и PyTorch. Это единственный инструмент из рассмотренных, не

поддерживающий формат ONNX. Для общения с сопроцессором DPU Vitis AI использует среду времени выполнения XRT [47].

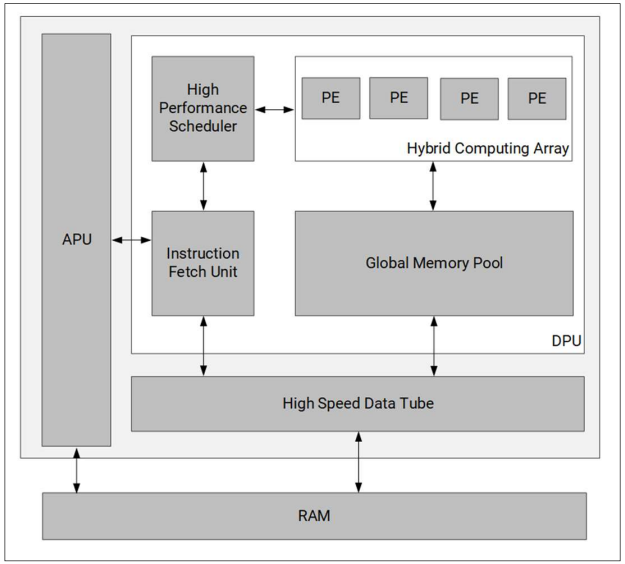

Рис. 3. Структура сопроцессора DPU
Fig. 3. DPU core structure

3.11.1 Deep Learning Processor Unit

Простейшее сопроцессорное ядро DPU для ПЛИС Zynq-7000 или Zynq UltraScale+ представлено на рис. 3 и состоит из следующих компонентов:

1) пула глобальной памяти (Global Memory Pool);
2) устройства выборки инструкций (Instruction Fetch Unit);
3) планировщика вычислений (High Performance Scheduler);
4) гибридного массива вычислителей (Hybrid Computing Array of Processing Engines);
5) внешних компонентов:
   a) блока управления приложением (Application Processing Unit, APU), контролирующего прерывания и передачу данных;
   b) высокоскоростного канала передачи данных (High Speed Data Tube).

Ядро DPU может быть сконфигурировано под определенную нейронную сеть с помощью выбора ширины входных и выходных каналов, функции активации и т.д. Для более продвинутых ПЛИС Xilinx существует ряд более мощных ядер DPU, ориентированных, например, на быструю обработку изображений.

3.12 FINN

FINN [48] – инструмент высокоуровневого синтеза моделей нейронных сетей, разработанный компанией Xilinx для ПЛИС этой же компании. Инструмент ориентирован на квантованные нейронные сети и принимает на вход модели в форматах PyTorch и ONNX. FINN оптимизирует операции с плавающей точкой и разделяет модель на синтезируемые и несинтезируемые слои. Синтезируемые слои реализуются с помощью библиотеки finn-hlslib [49]. Затем они оптимизируются, разделяются на сложнофункциональные блоки и

синтезируются в двоичный образ для ПЛИС с помощью инструмента Vivado. FINN также предоставляет среду времени выполнения, контролирующую процесс вычислений на ПЛИС.

4. Тестовые модели

4.1 MNIST-FC

MNIST-FC – полносвязная нейронная сеть, состоящая из одного слоя и распознающая рукописные цифры на изображениях набора MNIST. Имеет 784 входа (картинка 28×28 пикселей) и 10 выходов (каждый выход соответствует цифре от 0 до 9). Функция активации выходного слоя – *softmax*. Модель MNIST-FC была реализована и обучена на Keras и TensorFlow 1.6 (для инструмента LeFlow).

4.2 MNIST-CNN

MNIST-CNN – сверточная нейронная сеть, распознающая рукописные цифры из набора MNIST. Имеет такое же количество входов и выходов, как и модель MNIST-FC. Модель MNIST-CNN состоит из следующих слоев:

1) сверточный слой с функцией активации ReLu;
2) слой подвыборки;
3) полносвязный слой с функцией активации ReLu;
4) слой дропаута;
5) выходной слой без функции активации.

Модель MNIST-CNN была реализована и обучена на Keras и TensorFlow 1.6.

4.3 Синтетические примеры

Следующие три модели осуществляют умножение матриц и являются нагрузочными примерами для инструмента LeFlow (не осуществляют осмысленных вычислений). Они были реализованы только на TensorFlow 1.6.

1) MULT-0 – представляет собой умножение вектора размером N на матрицу размером N×N (по умолчанию, N=100)

2) MULT-10-R – синтетическая нейронная сеть из 10 полносвязных слоев. Каждый слой имеет 100 входов и 100 выходов и функцию активации *ReLu*. Обучение сети не производилось.

3) MULT-10-S – идентична MULT-10-R за исключением использования *сигмоида* в качестве функции активации последнего слоя.

5. Экспериментальная апробация

5.1 Эксперименты

Эксперименты проводились с помощью трех отладочных плат: Terasic DE10-Standard [50] и Terasic DE1-SoC [51], обе на базе ПЛИС Cyclone V, и платы Zybo Z7-20 [52] (система-на-кристалле Zynq-7000). Исходные модели нейронных сетей на Keras и TensorFlow исполнялись на ПК с процессором Intel Core i7-6700 3,4 ГГц (далее – CPU), 32 Гб RAM и ОС Ubuntu 20.04. Нейронные сети, оптимизированные с помощью инструмента TVM, исполнялись также на графическом процессоре NVIDIA GeForce GTX 770 (далее – GPU).

Главной целью экспериментов была реализация (и по возможности ускорение) моделей нейронных сетей на ПЛИС. Для этого были выбраны следующие инструменты: hls4ml, ONNC, Vitis AI, LeFlow и TVM (с сопроцессором VTA). Инструмент FINN не рассматривался из-за тесной интеграции с Vivado. Инструмент OpenVINO не участвовал в экспериментах из-

за ориентации на ПЛИС Intel Arria. Инструмент NNgen выдавал ошибки при обработке ONNX-моделей и был исключен из рассмотрения. До исполнения нейронной сети на ПЛИС удалось довести только два инструмента: LeFlow и TVM.

В процессе экспериментов замерялось непосредственно время исполнения нейронной сети, без времен обучения, компиляции и оптимизации. Также были получены показатели потребления ресурсов ПЛИС: количество использованных адаптивных логических модулей (ALM), процессоров обработки сигналов (DSP), регистров и блоков памяти, и максимальная частота синтезированных схем.

5.1.1 hls4ml

Инструмент успешно синтезировал C++-модели нейронных сетей MNIST-FC и MNIST-CNN, сохраненных в формате Keras H5. Затем была проведена безуспешная попытка синтезировать двоичные образы для ПЛИС с помощью инструмента Vivado 2020.1. Несмотря на то, что тестовые модели нейронных сетей являются простыми, оказалось, что количество связей между нейронами (представленными в виде циклов for в C++-модели) слишком велико для инструмента Vivado. При обработке модели MNIST-FC инструмент выдал сообщение, что не способен обработать подобный цикл for со слишком большим числом итераций. При обработке MNIST-CNN инструмент и вовсе завис на этапе оптимизации кода. Анализ кода MNIST-CNN показал, что в нем содержатся циклы с еще бóльшим количеством итерация, чем в коде MNIST-FC.

5.1.2 ONNC

Инструмент успешно синтезировал C++-модели MNIST-FC и MNIST-CNN, переданные ему в формате ONNX. Однако оказалось, что реализации слоев нейронных сетей скрыты во внутренней библиотеке ONNC, поэтому синтез двоичного образа для ПЛИС оказался невозможен.

Табл. 3. Параметры синтезированных моделей
Table 3. Synthesized Model Parameters

Параметр	**Модель**			
	MNIST-FC	***MULT-0***	***MULT-10-R***	***MULT-10-S***
Плата	DE1-SoC		DE10-Standard	
Размер RTL-модели, строк кода	5 240	1 252	6 011	11 490
Использование ALM, штук	5 331 (17%)	917 (3%)	4 598 (11%)	11 356 (27%)
Использование DSP, штук	1 (1%)	4 (5%)	31 (28%)	32 (29%)
Использование регистров, штук	7 311	1 554	6 842	16 277
Использование памяти, битов	278 986 (7%)	326 820 (8%)	3 213 220 (57%)	3 216 138 (57%)
Максимальная частота, МГц	127.39	122.55	96.08	93.63

5.1.3 Vitis AI

Эксперименты с Vitis AI не были проведены из-за отсутствия моделей сопроцессоров DPU в бесплатной версии Vivado. В качестве примера Xilinx предоставляет зашифрованный IP-блок на основе нескольких простых сопроцессоров DPU, однако он оказался слишком велик для имевшейся у нас в распоряжении отладочной платы Zybo Z7-20.

5.1.4 LeFlow

Инструмент синтезировал RTL-модели нейронной сети MNIST-FC и синтетических нейронных сетей, но не справился с моделью сверточной нейронной сети MNIST-CNN. Двоичные образы для ПЛИС были синтезированы с помощью инструмента Quartus [53] версии 20.1. В табл. 3 представлены параметры полученных RTL-моделей и соответствующих двоичных образов. Плата DE10-Standard была использована в случаях, когда синтезированная модель не помещалась на плате DE1-SoC.

Синтезированные RTL-модели были исполнены на симуляторе с целью получения их логического времени исполнения в тактах. Эти данные и значения максимальных частот были использованы для подсчета минимального времени исполнения нейронных сетей на ПЛИС. Также было измерено время исполнения исходных TensorFlow-моделей на CPU. Производительность полученных моделей представлена в табл. 4.

Табл. 4. Производительность синтезированных моделей
Table 4. Synthesized Model Performance

Параметр	Модель			
	MNIST-FC	***MULT-0***	***MULT-10-R***	***MULT-10-S***
Плата	DE1-SoC		DE10-Standard	
Логическое время исполнения, тактов	223 590	280 303	2 808 023	2 812 513
Время исполнения на CPU, мс	4	1	5	5
Время исполнения на ПЛИС, мс	1.76	2.29	29.23	30.03
Коэффициент ускорения (на ПЛИС)	2.28	0.44	0.17	0.17

5.1.5 TVM

Ранее было упомянуто, что сопроцессор VTA имеет версию для платы Terasic DE10-Nano, которая отличается от DE10-Standard моделью ПЛИС Cyclone V и некоторым периферийным оборудованием. Поэтому мы адаптировали ядро VTA для DE10-Standard путем изменения сборочных скриптов.

Табл. 5. Параметры двоичного образа сопроцессора VTA
Table 5. VTA Core Bitstream Parameters

Параметр	Значение
Плата	DE10-Standard
Использование ALM, штук	20 311 (48%)
Использование DSP, штук	0 (0%)
Использование регистров, штук	19 226
Использование памяти, битов	3 905 016 (69%)
Максимальная частота, МГц	137.85

Более сложной задачей было установить среду времени выполнения TVM на нашу плату, так как стандартные версии ОС Linux не подходили для сборки среды. Для этого был собран образ ОС Linux на основе ядра версии 4.14.130 [54] и файловой системы Ubuntu Base [55]

версии 18.04.5, а также найден подходящий драйвер [56] резервирования непрерывной области памяти (Contiguous Memory Allocator, CMA).

Двоичный образ сопроцессора VTA был синтезирован с помощью инструмента Quartus 18.1. Параметры образа представлены в табл. 5.

Инструмент TVM скомпилировал, оптимизировал и исполнил на ПЛИС обе модели нейронных сетей MNIST-FC и MNIST-CNN. Примеры, разработанные на TensorFlow 1.6, не были обработаны TVM, так как он не поддерживает столь старую версию TensorFlow. Во время экспериментов измерялись времена исполнения оптимизированных моделей на CPU, GPU и VTA. Время исполнения исходных Keras-моделей было также измерено. Модель MNIST-CNN была квантована и исполнена с помощью средств TVM. Производительность полученных моделей представлена в табл. 6. Ускорение вычислялось в сравнении с исходной Keras-моделью.

Табл. 6. Производительность сопроцессора VTA
Table 6. VTA Core Performance

Параметр		Модель		
		MNIST-FC	***MNIST-CNN***	***MNIST-CNN (с квантованием)***
Время исполнения на CPU, мс	Keras	0.38	2.55	н/д
	TVM	0.0037	0.8	0.44
Время исполнения на GPU, мс	TVM	0.02	0.17	Ошибка CUDA
Время исполнения на VTA, мс		0.10	105.0	33.3
Коэффициент ускорения (на сопроцессоре)	Keras	3.8	0.024	0.08
	TVM, CPU	0.037	0.008	0.014
	TVM, GPU	0.020	0.0016	н/д

5.2 Анализ

Эксперименты показали, что большинство открытых инструментов способны обрабатывать модели нейронных сетей. Однако некоторые из них не могут быть использованы для исполнения нейронных сетей на ПЛИС (ONNC, hls4ml). Некоторые инструменты зависят от коммерческих продуктов (hls4ml, Vitis AI и др.). Только TVM и LeFlow оказались способны исполнить (или синтезировать) тестовые модели на ПЛИС. Но LeFlow не синтезировал сверточную нейронную сеть MNIST-CNN.

Исполнение моделей нейронных сетей на ПЛИС показало неоднозначные результаты. Простая полносвязная сеть MNIST-FC была ускорена по сравнению с исходной Keras/TensorFlow-моделью. С другой стороны, другие модели исполнялись существенно медленнее даже по сравнению с исходными моделями. Оптимизированные с помощью TVM модели показали лучшее время исполнения на CPU (для MNIST-FC) и GPU (для MNIST-CNN). Модель MNIST-CNN, исполненная на VTA, оказалась в сотни раз медленнее, чем исполненная на CPU или GPU. Квантование и упаковка улучшили производительность модели на VTA, но несущественно. Что касается LeFlow, производительность синтезированных схем почти полностью зависит от инструмента LegUp.

Слабые результаты, показанные сопроцессором VTA для сверточной нейронной сети, можно объяснить неудачно подобранными стандартными настройками ядра, плохо совместимыми с моделью нейронной сети. Эта тема требует дальнейшего исследования. Также к недостаткам сопроцессора VTA можно отнести его небольшие размеры и, как следствие, необходимость частой пересылки больших объемов данных между микропроцессором и ПЛИС через память

системы-на-кристалле. Решением может быть увеличение размеров сопроцессора и использование ПЛИС с бóльшим числом логических элементов.

6. Заключение

В данной работе были рассмотрены популярные форматы представления нейронных сетей и открытые инструменты для их исполнения и ускорения. Несколько инструментов было исследовано экспериментально с помощью простых моделей нейронных сетей. Результаты проведенных экспериментов показали, что открытые инструменты способны обрабатывать различные нейронные сети и ускорять их на микро- и графических процессорах. Однако производительность на ПЛИС обычно хуже, за исключением самых простых примеров.

Проведенные эксперименты показали, что применение открытых инструментов вместе с небольшими ПЛИС целесообразно только в системах с жесткими ограничениями на энергопотребление (например, встраиваемых системах) и мягкими требованиями по производительности. Использование открытых инструментов для ускорения нейронных сетей на больших ПЛИС требует дальнейшего исследования.

Дальнейшие исследования возможны и в других областях: разработке специализированных архитектур ускорителей, методах оптимизации нейронных сетей, использованию гетерогенных систем для распределения специфических вычислений, использованию специальных типов нейронных сетей и т.д.

Список литературы / References

[1] TensorFlow framework. Available at: https://www.tensorflow.org, accessed 21.10.2021.
[2] PyTorch framework. Available at: https://pytorch.org, accessed 21.10.2021.
[3] Caffe framework. Available at: https://caffe.berkeleyvision.org, accessed 21.10.2021.
[4] ONNX format. Available at: https://onnx.ai, accessed 21.10.2021.
[5] Lu Y., Gong L. et al. A high-performance FPGA accelerator for sparse neural networks: work-in-progress. In Proc. of the 2017 International Conference on Compilers, Architectures and Synthesis for Embedded Systems Companion, 2017, pp. 1-2.
[6] Sigurbergsson B., Hogervorst T. et al. Sparstition: A Partitioning Scheme for Large-Scale Sparse Matrix Vector Multiplication on FPGA. In Proc. of the IEEE 30th International Conference on Application-specific Systems, Architectures and Processors, 2019, pp. 51-58.
[7] Yang L., He Z., Fan D. A Fully Onchip Binarized Convolutional Neural Network FPGA Implementation with Accurate Inference. In Proc. of the International Symposium on Low Power Electronics and Design, 2018, pp. 1-6.
[8] Chi C., Jiang J. R. Logic Synthesis of Binarized Neural Networks for Efficient Circuit Implementation. In Proc. of the IEEE/ACM International Conference on Computer-Aided Design, 2018, pp. 1-7.
[9] Zhuge C., Liu X. et al. Face Recognition with Hybrid Efficient Convolution Algorithms on FPGAs. In Proc. of the Great Lakes Symposium on VLSI, 2018, pp. 123-128.
[10] Xiao Q., Liang Y. et al. Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs. In Proc. of the 54th Annual Design Automation Conference, 2017, pp. 1-6.
[11] Lu L., Liang Y. SpWA: an efficient sparse Winograd convolutional neural networks accelerator on FPGAs. In Proceedings of the 55th Annual Design Automation Conference, 2018, pp. 1-6.
[12] Alwani M., Chen H. et al. Fused-layer CNN accelerators. In Proc. of the 49th Annual International Symposium on Microarchitecture, 2016, pp. 1-12.
[13] Samragh M., Javaheripi M., Koushanfar F.F. EncoDeep: Realizing Bit-flexible Encoding for Deep Neural Networks. ACM Transactions on Embedded Computing Systems, vol. 19, issue 6, 2020, Article 43, 29 p.
[14] Ding C., Wang S. et al. REQ-YOLO: A Resource-Aware, Efficient Quantization Framework for Object Detection on FPGAs. In Proc. of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019, pp. 33-42.
[15] Karki A., Keshava C. P. et al. Detailed Characterization of Deep Neural Networks on GPUs and FPGAs. In Proc.of the 12th Workshop on General Purpose Processing Using GPUs, 2019, pp. 12-21.
[16] Mittal S., Umesh S. A survey on hardware accelerators and optimization techniques for RNNs. Journal of Systems Architecture, vol. 112, 2020, article no. 101839.

[17] Kouris A., Venieris S. I., Bouganis C.-S. A throughput-latency co-optimised cascade of convolutional neural network classifiers. In Proc. of the 23rd Conference on Design, Automation and Test in Europe, 2020, pp. 1656-1661.
[18] Lin X., Yin S. et al. LCP: a layer clusters paralleling mapping method for accelerating inception and residual networks on FPGA. In Proc. of the 55th Annual Design Automation Conference, 2018, pp. 1-6.
[19] Keras framework. Available at: https://keras.io, accessed 21.10.2021.
[20] Bücker H. M., Corliss G. et al., editors. Automatic Differentiation: Applications, Theory, and Implementations. Lecture Notes in Computational Science and Engineering, vol. 50, Springer, 2006, 361 p.
[21] Caffe2 framework. Available at: https://caffe2.ai, accessed 21.10.2021.
[22] CNTK framework. Available at: https://github.com/Microsoft/CNTK, accessed 21.10.2021.
[23] CoreML framework. Available at: https://coremltools.readme.io, accessed 21.10.2021.
[24] MXNet framework. Available at: https://mxnet.apache.org, accessed 21.10.2021.
[25] Theano framework. Available at: https://github.com/Theano/Theano, accessed 21.10.2021
[26] Aesara library. Available at: https://github.com/aesara-devs/aesara, accessed 21.10.2021
[27] LegUp 4.0 high-level synthesis tool. Available at: http://legup.eecg.utoronto.ca, accessed 30.12.2020.
[28] Vivado Design Suite. Available at: https://www.xilinx.com/products/design-tools/vivado.html, accessed 21.10.2021.
[29] OpenVINO toolkit. Available at: https://github.com/openvinotoolkit/openvino, accessed 21.10.2021.
[30] Kaldi Speech Recognition Toolkit. Available at: https://kaldi-asr.org, accessed 21.10.2021.
[31] PlaidML framework. Available at: https://github.com/plaidml/plaidml, accessed 21.10.2021.
[32] nGraph compiler stack. Available at: https://github.com/NervanaSystems/ngraph, accessed 21.10.2021.
[33] MACE framework. Available at: https://github.com/XiaoMi/mace, accessed 21.10.2021.
[34] Apache TVM ML compiler framework. Available at: https://tvm.apache.org, accessed 21.10.2021.
[35] VTA: Deep Learning Accelerator Stack. Available at: https://tvm.apache.org/docs/vta/index.html
[36] XLA neural network compiler. Available at: https://www.tensorflow.org/xla, accessed 21.10.2021.
[37] The Julia programming language. Available at: https://julialang.org, accessed 21.10.2021.
[38] JAX differentiation library. Available at: https://github.com/google/jax, accessed 21.10.2021.
[39] Nx multidimensional array representation library. Available at: https://github.com/elixir-nx/nx, accessed 21.10.2021.
[40] LeFlow tool-flow. Available at: https://github.com/danielholanda/LeFlow, accessed 21.10.2021.
[41] Glow ML compiler. Available at: https://ai.facebook.com/tools/glow, accessed 21.10.2021.
[42] hls4ml tool. Available at: https://github.com/fastmachinelearning/hls4ml, accessed 21.10.2021.
[43] Kreinar E. Rfnoc Neural Network Library Using Vivado HLS. In Proc. of the GNU Radio Conference, 2017, 7 p.
[44] NNgen hardware synthesis compiler. Available at: https://github.com/NNgen/nngen, accessed 21.10.2021.
[45] ONNC compilation framework. Available at: https://onnc.ai, accessed 21.10.2021.
[46] Xilinx Vitis AI development stack for AI inference. Available at: https://github.com/Xilinx/Vitis-AI, accessed 21.10.2021.
[47] Xilinx runtime for FPGA. Available at: https://github.com/Xilinx/XRT, accessed 21.10.2021.
[48] FINN framework. Available at: https://xilinx.github.io/finn, accessed 21.10.2021.
[49] Vivado HLS library for FINN. Available at: https://github.com/Xilinx/finn-hlslib, accessed 21.10.2021.
[50] Terasic DE10-Standard board. Available at: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=1081. accessed 21.10.2021.
[51] Terasic DE1-SoC board. Available at: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836, accessed 21.10.2021.
[52] Zybo Z7-20 board. Available: https://www.xilinx.com/products/boards-and-kits/1-pukio3.html, accessed 21.10.2021.
[53] Quartus Prime Software Suite. Available at: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html, accessed 21.10.2021.
[54] Linux development repository for socfpga. Available at: https://github.com/altera-opensource/linux-socfpga/tree/socfpga-4.14.130-ltsi, accessed 21.10.2021.
[55] Ubuntu Base rootfs. Available at: https://wiki.ubuntu.com/Base, accessed 21.10.2021.
[56] Linux drivers (modules) for Field Programmable Systems on Chip. Available at: http://git.edi.lv/rihards.novickis/FPSoC_Linux_drivers, accessed 21.10.2021.

Информация об авторах / Information about authors

Михаил Сергеевич ЛЕБЕДЕВ – сотрудник ИСП РАН, а также научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова. Сфера научных интересов: высокоуровневый синтез, методы исследования проектных альтернатив, нейронные сети, цифровая аппаратура, методы верификации цифровой аппаратуры.

Mikhail Sergeyevich LEBEDEV is a specialist 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.

Павел Николаевич БЕЛЕЦКИЙ – сотрудник ИСП РАН. Сфера научных интересов: высокоуровневый синтез, искусственные нейронные сети, ускорение искусственных нейронных сетей, математическое моделирование биологических систем.

Pavel Nikolaevich BELECKY is a specialist of ISP RAS. Research interests: high-level synthesis, artificial neural networks, acceleration of artificial neural networks, mathematical modeling of biological systems.