Koios 2.0: Open-Source Deep Learning Benchmarks for FPGA Architecture and CAD Research

Aman Arora[1], Andrew Boutros[2], Seyed Alireza Damghani[3], Karan Mathur[1], Vedant Mohanty[1], Tanmay Anand[1], Mohamed A. Elgammal[2], Kenneth B. Kent[3], Vaughn Betz[2], Lizy K. John[1]
[1]*The University of Texas at Austin* [2]*University of Toronto* [3]*University of New Brunswick*
E-mail: aman.kbm@utexas.edu

***Abstract*—With the prevalence of deep learning (DL) in many applications, researchers are investigating different ways of optimizing FPGA architecture and CAD to achieve better quality-of-results (QoR) on DL-based workloads. In this optimization process, benchmark circuits are an essential component; the QoR achieved on a set of benchmarks is the main driver for architecture and CAD design choices. However, current academic benchmark suites are inadequate, as they do not capture any designs from the DL domain. This work presents the second version of our suite of DL acceleration benchmark circuits for FPGA architecture and CAD research, called Koios. This suite of 40 circuits covers a wide variety of accelerated neural networks, design sizes, implementation styles, abstraction levels, and numerical precisions. These benchmarks include 32 DL designs and 8 synthetic (proxy) benchmarks. The Koios benchmarks are larger, more data parallel, more heterogeneous, more deeply pipelined, and utilize more FPGA architectural features compared to existing open-source benchmarks. This enables researchers to pinpoint architectural inefficiencies for this class of workloads and optimize CAD tools on more representative benchmarks that stress the CAD algorithms in different ways. In this paper, we describe the Koios designs, compare their characteristics to prior FPGA benchmark suites, and present results of running them through the Verilog-to-Routing (VTR) flow using a recent FPGA architecture model. Finally, we present case studies showing how exploration of DL-optimized FPGA architecture and CAD algorithms can be performed using our new benchmark suite.**

***Index Terms*—FPGA, benchmarks, Deep Learning, VTR, Koios, Titan**

I. Introduction

With compute and data intensive deep learning (DL) becoming a major component of many applications, specialized hardware acceleration of such workloads has become commonplace. More recently, field-programmable gate arrays (FPGAs) have been shown to deliver state-of-the-art performance when accelerating different DL workloads because of their massive parallelism, flexibility and energy efficiency [1], [2]. With new DL use cases emerging faster than ever, FPGAs are also starting to adapt. This includes the emergence of DL-optimized FPGA fabrics [3], the integration of FPGAs with specialized DL accelerators [4], [5], and also tuning FPGA CAD tools to the properties of these workloads [6].

As shown in Figure 1, the process of development of novel FPGA architectures and CAD algorithms involves: (1) a set of benchmarks written in a hardware description language or synthesized using high-level synthesis, (2) an architecture model that captures the organization of FPGA blocks and routing architecture as well as area/timing/power models from circuit-level implementations, and (3) a CAD flow that implements the given benchmarks on a target FPGA architecture [7]. Although most research efforts in the FPGA community are focused on architecture and CAD, benchmarks also play a crucial role in this flow. The quality-of-results (QoR) achieved on a specific set of benchmarks is the main driver for architecture and CAD design choices. As a result, it is essential that these benchmarks capture the markets and application domains targeted by the candidate FPGA architecture. Using an unrepresentative set of benchmarks means optimizing for the wrong targets.

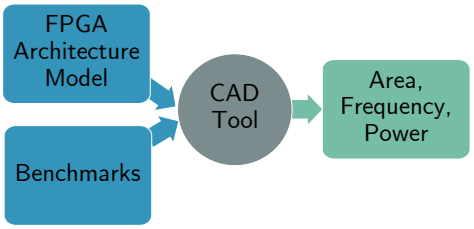


Fig. 1: FPGA architecture and CAD algorithm exploration requires a set of representative benchmarks

Existing open-source benchmark suites do not focus on (or even capture any) benchmarks from the increasingly important DL domain. Therefore, it impedes the development of architecture and CAD optimizations for DL-targeted FPGAs, since researchers have to first implement their own benchmarks. This limits any research efforts in this direction to only individual isolated ones, and makes it virtually impossible to have meaningful comparisons between different ideas across the FPGA research community. Our work addresses this by presenting Koios 2.0[1], an open-source benchmark suite of DL acceleration benchmark circuits. These benchmarks can be used by FPGA researchers for exploring FPGA architectures and tuning FPGA CAD algorithms. This suite consists of 40 benchmarks that capture a wide variety of accelerated neural networks, design sizes, numerical precisions, and circuit characteristics. To maximize the utility of these benchmarks, we made them compatible with the Verilog-to-Routing (VTR) flow [8], which is the most widely-used FPGA architecture and CAD research framework. Researchers can use these benchmarks seamlessly with VTR and, with minor modifications, can also use them with other toolchains.

[1]*Koios* (also written as *Coeus*) is the Titan of intelligence in Greek mythology. Unlike the Titan benchmarks, our suite focuses on deep learning.

The Koios benchmarks are representative of modern DL workloads; many of them are re-created from prior works and some are replicas of industrial benchmarks. In addition to being more pipelined and heavily using FPGA hard blocks[2], these benchmarks have higher usage of structures like wide buses, large reduction trees, hard block dedicated cascade routing and large fanouts. This makes the Koios benchmarks better suited for DL-targeted FPGA architecture exploration than other non-DL benchmark suites.

In this paper, we introduce Koios 2.0, an enhanced version of the Koios suite [9], and make the following contributions:

- Extending the Koios suite, which originally contained 19 benchmarks, with 13 new DL application circuits.
- Introducing a framework for generating synthetic/proxy benchmarks with specific circuit characteristics, and extending the Koios suite with 8 proxy benchmarks.
- Showcasing a new synthesis frontend (Yosys+Odin) in the VTR flow that significantly expands the Verilog and SystemVerilog syntax coverage of VTR.
- Presenting the results of running our benchmarks through VTR using an FPGA architecture model that we develop to capture complex hard blocks typical of recent FPGAs.
- Comparing circuit statistics to the VTR and Titan benchmarks to highlight the added value of Koios.
- Performing a QoR comparison of VTR with a commercial tool chain (Intel Quartus) for the Koios benchmarks on the Stratix-IV FPGA architecture.
- Describing example case studies that use these benchmarks to explore architecture and CAD optimizations for DL.

II. Related Work

A. FPGA Benchmark Suites

Several benchmark suites have been curated and used by FPGA architecture and CAD researchers over the past three decades. Table I provides an overview of the features of the different suites. The classic MCNC20 (the twenty largest MCNC) benchmarks [10] are extremely small (less than 10K LUTs) and simple designs that do not use any FPGA hard blocks. While these designs were used in many early CAD and architecture studies such as [11], they are no longer very representative of modern FPGA use-cases. The UMass RCG HDL Benchmark Collection [12] has somewhat larger designs of up to $14,000$ look-up tables (LUTs) mostly representing digital signal processing (DSP) applications. However, this suite does not target an open-source FPGA framework, which limits its use in architecture and CAD studies as they generally need modifiable and retargetable CAD tools. The Groundhog benchmarks [13] are intended to be architecture independent; they work with academic tool flows and are targeted towards evaluation of power consumption of FPGAs for mobile computing applications. However, only two of the benchmarks have HDL realizations (and hence can be run through an implementation CAD flow) and both are very small (under $1,000$ primitives). ERCBench [14] consists of hybrid hardware/software applications. The designs in this suite are from the multimedia, wireless communications and cryptography domains and it contains some medium size designs (up to $65,000$ LUTs). They do not contain DL benchmarks, and do not readily work with academic (open source) FPGA tools.

TABLE I: Comparing FPGA benchmark suites

Benchmark Suite	Max. primitives per design	Use of Hard Blocks	Open Source CAD	Captures DL Domain
MCNC20 [10]	10K	×	✓	×
UMass RCG [12]	14K	✓	×	×
Groundhog [13]	1K	✓	✓	×
ERCBench [14]	65K	✓	×	×
VTR [8]	165K	✓	✓	×
Titan [15]	1.8M	✓	×	×
Koios 2.0 (This work)	1.6M	✓	✓	✓

VTR [8] has a suite of Verilog benchmarks as well. These VTR benchmarks vary from small (321 netlist primitives) to medium-sized designs ($165,809$ primitives) and they include applications from several domains including image processing, soft processors and arithmetic. The Titan benchmark suite [15] contains modern heterogeneous large designs (90K to 1.8M netlist primitives); these are HDL benchmarks (some of which were generated from high-level synthesis) that are provided as both as the source HDL and BLIF [16] format netlists that can be input to VPR [8], [11] (VPR is the tool that performs packing, placement and routing in the VTR flow). However, they target a hybrid CAD flow that is architecture-specific as logic synthesis is performed using the Intel Quartus tool only for the Stratix-IV architecture. In contrast to all existing suites, Koios is the only one that provides large, heterogeneous, architecture-agnostic benchmarks that work with a completely open-source flow, and focuses on the increasingly important DL domain.

The Koios suite contains only Verilog benchmarks. The focus of these benchmarks is the exploration of new FPGA architectures and CAD algorithms (i.e. HDL to physical design). There are other FPGA benchmarks written in C or OpenCL such as the Rosetta [17] and Spector [18] suites. The focus of these benchmarks is the FPGA HLS flow (i.e. high-level language to HDL).

B. DL-Optimized FPGAs

Recently, FPGA vendors have released products with new DL-targeted features to cater to the ever-growing demands of this domain. For example, the Xilinx Versal ACAP [5] added a 2D mesh of specialized vector processors connected by streaming interconnect, and Intel's Stratix 10 NX devices integrated in-fabric AI tensor blocks [3]. In addition, the Achronix Speedster7t FPGA [19] has embedded machine learning processor blocks that tightly couple memory and compute for DL. For architecture exploration, FPGA vendors typically use proprietary customer designs or internal benchmarks that are not accessible to the research community.

There have also been a number of academic research proposals to optimize FPGA architectures for DL. Eldafrawy

[2]A hard block on an FPGA is a fixed function block that is embedded into the silicon. It provides higher level functionality and performance than building those functions from logic blocks. Examples of hard blocks are multipliers (DSP blocks) and memory (Block RAMs).

et al. [20] proposed several enhancements to logic blocks to increase the density of multiply-accumulate (MAC) operations implemented in the FPGA's soft logic. They used simple multiplier/MAC and 4×4 matrix multiplication microbenchmarks to evaluate their proposed ideas. In [21], [22], the authors explored enhancing DSP blocks by efficiently supporting low precision multiplications and adding register files inside DSPs for storage. For these studies, the authors designed their own benchmarks to evaluate their ideas. Arora et al. [23] also proposed adding Tensor slices in FPGAs. Again, they use their own designs – a TPU-like overlay and several microbenchmarks – for their evaluation. The authors of [24] and [25], propose adding compute capabilities into block RAMs (BRAMs) to increase the computational throughput of FPGAs and reduce interconnect usage. They use custom benchmarks to illustrate how these enhancements improve the performance of DL applications.

All these investigations evaluate architecture ideas primarily on microbenchmarks, with some also including a small number of larger designs. The exact microbenchmarks and applications used also vary between studies, making comparisons of the gains achieved by different works difficult. An open-source DL benchmark suite is needed to enable evaluation of architecture enhancements across a wider range of use cases and to facilitate comparisons across research works.

III. The Koios 2.0 Benchmark Suite

A. Overview

The Koios 2.0 benchmark suite is a DL-specific benchmark suite for FPGA research. It consists of 40 benchmarks covering a diverse representative space, coming from various applications within the DL domain. Table II provides an overview of the benchmarks and their properties. These benchmarks are completely open-source, and we provide them as both Verilog HDL source codes and BLIF netlists.

B. Diversity and Representativeness

The Koios benchmarks cover a wide variety of design sizes, implementation styles, target neural networks, acceleration paradigms, numerical precisions, and circuit properties.

- **Design Size**: The smallest design has $12,097$ netlist primitives while the largest has $1,608,867$. Any latch, gate or hard block resulting from logic synthesis counts as a netlist primitive. Some benchmarks, such as `clstm_like`, `dla_like`, `tpu_like`, have multiple size variants (i.e. small, medium, large). In these cases, the size indicates the parallelism factor used in the design. Bigger designs create a more challenging optimization problem for the CAD tools, while smaller ones have faster compilation time suitable for early-stage experiments.
- **Implementation Style**: Although all the designs in the benchmark suite are provided to users in the form of Verilog HDL implementations, some were originally implemented in RTL while others were automatically generated from higher-level language descriptions using high-level synthesis (HLS) tools. HLS-generated designs typically have specific design characteristics that are not very common in hand-coded RTL designs, such as widely distributed control signals and complex state machines.
- **Target Neural Network**: Our benchmarks cover all major classes of neural networks. These include: multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), and reinforcement learning (RL). These different classes have different compute and memory requirements, which reflects on the resource breakdown and routing patterns of their corresponding benchmark circuits. Some designs are also generic and can be used to accelerate any type of network.
- **Acceleration Paradigm**: FPGAs are used for acceleration of DL workloads in different ways. One way is to design a flexible software-programmable overlay architecture that can execute different DL models without the need to reprogram the FPGA with a new bitstream similar to the Microsoft Brainwave [31] architecture. These designs tend to have instruction decoders and more complicated control logic to enable this level of flexibility. In other cases, a custom network-specific dataflow architecture is mapped to an FPGA to maximize efficiency similar to [1]. The control logic of these circuits is usually hard-coded and implemented as relatively simple state machines. Another approach is to implement layer-specific accelerators that are invoked by software running on a host or an embedded CPU. These circuits are mostly streaming-style datapaths with simple or even no control paths. Our benchmark suite contains designs from all three acceleration paradigms.
- **Numerical Precisions**: One of the main advantages of using FPGAs to accelerate DL workloads is the ability to design hardware for custom numerical precisions, which is a commonly used technique in accelerating DL workloads [42]. The designs in our suite use various precisions, including: binary (`bin`), different fixed point types `int4/8/16/18/32`, brain floating point (`bfloat16` [43]), IEEE half-precision floating point (`fp16`), and block floating point (`bfp11` [31]). This diversity is useful for exploring new DSP block architectures and different hard arithmetic circuitry.
- **Circuit Properties**: Our benchmarks have varying circuit styles that can exercise different components of the CAD tools in different ways. For example, regular structures like systolic arrays can be used for optimizing placement algorithms, large reduction trees can form local routing congestion that stress the routing algorithms, long cascades (or chains) of hard blocks impose harder placement constraints, etc. The benchmarks are also highly heterogeneous (i.e. use different types of FPGA resources) with varying degrees. They utilize a large number of DSP blocks and BRAMs. DSPs are often used to form dot product units and memory structures like double-buffered RAMs and FIFOs are commonly used to store on-chip weights and activations.

C. Curating the benchmark suite

The designs in the benchmark suite are chosen keeping representativeness and diversity in mind. These designs are implemented (either handcoded or script generated or using

TABLE II: The Koios 2.0 Benchmarks (in decreasing order of number of netlist primitives)

Benchmark	Description	Implementation	Network	Precision	Acc. Paradigm	2D Systolic	Winograd/FFT	Reduction	Buffers	DSP usage	Cent. buffers	Based on	Other Properties
dla_like (S/M/L)	Intel-DLA-like accelerator	RTL	CNN[2]	int8/16	Overlay		✓		✓[3]	✓[4]	✓	[26] [27]	Daisy chain
clstm_like (S/M/L)	CLSTM-like accelerator	RTL	RNN	int18	Overlay		✓		✓[3]	✓		[28]	Circular compression
deepfreeze	ARM FixyNN design	RTL	CNN	int4	Layer			✓	✓	✓		[29]	Hardcoded weights
tdarknet_like (S/L)	Accelerator for Tiny Darknet	HLS	CNN[12]	fp16	Custom				✓[3]	✓		[30]	Fused layer pairs
bwave_like	Microsoft-Brainwave-like design	RTL	Any	int8, bfp11	Overlay			✓	✓	✓[4]		[31]	Mat-vec mult unit
lstm	LSTM engine	RTL	RNN	int16	Layer			✓	✓	✓			Streaming dataflow
bnn	4-layer binary neural network	HLS	MLP[1]	binary	Custom					✓		[32]	int16 act/norm
lenet	Accelerator for LeNet-5	HLS	CNN	int8	Custom				✓	✓		[33]	5x5 conv layers
dnnweaver	DNNWeaver accelerator	RTL	Any	int8	Overlay	✓			✓[3]	✓	✓	[34]	DDR and PCIe intf
tpu_like.ws (S/L)	Google-TPU-v1-like accelerator	RTL	Any[12]	int8	Overlay	✓			✓	✓	✓	[35]	Weight stationary MMU
tpu_like.os (S/L)	Google-TPU-v1-like accelerator	RTL	Any[12]	int8	Overlay	✓			✓	✓	✓	[35]	Output stationary MMU
gemm_layer	Matrix multiplication engine	RTL	MLP	bfloat16	Layer	✓				✓	✓		AXI interface
attention_layer	Transformer self-attention layer	RTL	RNN	int16	Layer			✓	✓[3]	✓		[36]	GEMV based
conv_layer	GEMM based convolution	RTL	CNN	int16	Layer	✓			✓	✓	✓		3x3 filters
robot_rl	Robot+maze application	RTL	RL	int8/16/32	Custom				✓	✓	✓	[37] [38]	Q-learning algo
reduction_layer	Add/max/min reduction tree	RTL	Any	int16	Layer			✓	✓		✓		Reduces 128 inputs
spmv	Sparse matrix vector multiplication	RTL	MLP	int8	Layer				✓	✓	✓	[39] [40]	COO sparsity enc.
eltwise_layer	Matrix elementwise add/sub/mult	RTL	Any	bfloat16	Layer				✓	✓	✓		Broadcast heavy
softmax	Softmax classification layer	RTL	Any	fp16	Layer			✓		✓		[41]	LUT based exp/log
conv_layer_hls	Sliding window convolution	HLS	CNN	fp16	Layer				✓	✓			1x1 filters
proxy	Proxy benchmarks	RTL	-	-	-	-	-	-	-	-	-	Sec III-D	-

[1] Has Normalization layer [2] Has pooling layer [3] Uses double buffering [4] Has DSP cascade chains

HLS) and tested using commercial FPGA tools for ease of development and debugging. Then, we performed many modifications to these designs to ensure their compatibility with the VTR flow. Vendor-specific and architecture-specific IP cores (e.g. floating point adders and multipliers, RAM macros) were replaced with ones that are compatible with VTR and the FPGA architecture file used for our experiments. This process was especially challenging for the designs generated from HLS tools, which tend to be non-human-readable in many cases. After that, various experiments (described later in this paper) were run to ensure the suitability of these benchmarks.

D. Proxy benchmarks

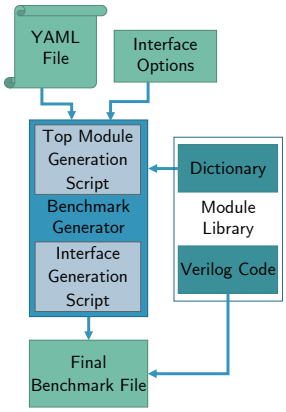
In the first version of Koios [9], there were 19 benchmarks and in this work, we added 13 more designs. However, having a larger set of benchmark circuits is desirable for most FPGA architecture and CAD research. Obtaining real world designs and curating them to be used as FPGA benchmarks is a tedious process as it requires re-creating designs that are not publicly available or modifying existing ones to be compatible with open source CAD tools. Hence, deriving inspiration from other fields [44], [45], we create a framework for generating synthetic DL benchmark circuits. The synthetic benchmarks generated by this framework have similar properties and circuit compositions to real DL benchmarks as described in the previous section. Since these benchmarks can be used as proxies of real DL designs for FPGA architecture and CAD research, we refer to them as *proxy benchmarks*. Unlike the other benchmarks in Koios, the generated proxy benchmarks are not functional DL accelerators – they instead mimic the composition of key components of DL accelerators. We perform statistical analysis on the properties of real designs and synthetic designs generated from this framework, and compare them in Section IV-D.

Proxy benchmarks are generated using design components that are commonly present in real DL designs. We extracted and parameterized different components from the existing benchmarks and designed new ones to create a library of modules that can be used in the generation of proxy benchmarks, as listed in Table III. This library can be easily extended to increase the diversity of the generated proxy benchmarks. In addition to the Verilog implementation of these components, the library also contains a Python dictionary of the various components along with their properties (e.g. size, precision, width) and the resource usage of each module for the FPGA architecture we use for evaluation. Fig. 2a shows how the proxy benchmark generation framework works. The benchmark generator takes as input a YAML file which specifies the graph structure the user desires (i.e. the specific hardware components and the connections between them). A snippet from a sample YAML file is shown in Fig. 2b. For each component, its specific parameters (e.g. type, size, precision) are also specified in the YAML file. The generator goes through the graph structure described by the user, instantiates the corresponding components in the top-level module, and automatically generates the interconnections between them to generate the Verilog file of the proxy benchmark.

Since the YAML input file specifies components connected to each other regardless of the number of output and input bits of each component, the generator inserts interfacing logic between the component instances. For example, if the YAML file specifies that component A with 40 output bits feeds component B that has 20 input bits, some interface logic needs to be generated that can enable connecting 40 signals to 20 signals. There are three cases that can arise:

TABLE III: Circuit components used to generate proxy benchmarks

Type	Properties
Adder Tree	Adder tree levels {3,4,5}, Precision {16, 8, 4, fp16}
DSP Chain	# DSPs {2,3,4}, Precision {16, fp16}
Systolic Array	Array size {4x4, 8x8}, Precision {4, 8, fp16}
Activations	# of logical LUTs {32}, Precision {8, 16}
Dot Product	Dot product length {10}, Precision {8, bf16}
RAM	Depth {2048, 4096}, Width {40, 60}, # Ports {1, 2}
Double Buffer	Depth {2048, 4096}, Widths {40, 60}
FIFO	Depth {256, 512}, Widths {40, 60}



(a) Flow overview

```
fifo1:
  type: "fifo"
  size: 256
  precision: 40
  number: 4
  inputs: [top]
  outputs: [dsp_ch1]
dsp_ch1:
  type: "dsp_chain"
  size: 4
  precision: 18
  number: 8
  inputs: [fifo1, add_tree2]
  outputs: [activ1]
activ1:
  type: "sigmoid"
  size: 16
  precision: 16
  number: 1
  inputs: [dsp_ch1]
  outputs: [sarray2,dbram2]
```

(b) YAML file snippet

Fig. 2: Proxy benchmark generation

1) The input bits are equal to the output bits and can be directly connected.
2) The input bits are less than the output bits. In this case, input bits are fanned out to match the number of outputs.
3) The input bits are greater than output bits. In this case, a reduction of bits is performed by inserting reduction trees of logical operations (e.g. xor/and/or gates, 2:1 multiplexers). The user can specify the mix of gates and multiplexers to be used via command line options ("Interface options" in Fig. 2a).

The interface logic always adds a register stage between components. Note that these choices are also governed by circuit properties observed in real world designs; deep pipelines, high fanouts and reduction trees are common in DL designs.

The proxy benchmark generator also generates statistics, such as the number of I/Os, the expected number of logic blocks, DSPs and BRAMs used by the generated benchmark. Comparing these numbers with the numbers obtained after running the generated benchmark through the VTR flow can be useful for verifying the validity of the generated benchmark.

Currently, the user has to specify the exact structure of the proxy benchmark in the input YAML file. However, an enhancement to this framework where the user only needs to specify an approximate mixture of components they desire is work-in-progress. A YAML file can then be automatically generated and passed as an input to the existing framework, which enables easier and faster generation of proxy benchmarks from circuit properties. For the Koios 2.0 suite, we have generated 8 proxy benchmarks of varying sizes ($14-43K$ netlist primitives) and containing different mixes of components from our module library. We have open-sourced the generator and the YAML files of the 8 benchmarks, so a user can generate more designs, if required.

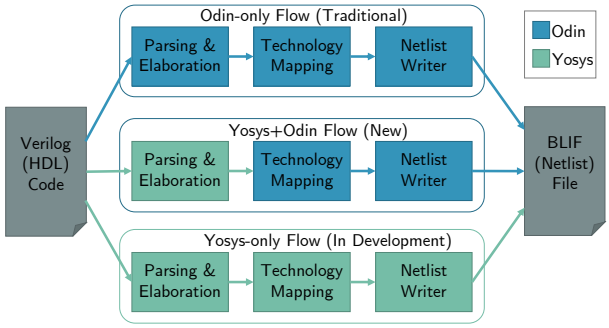


Fig. 3: Various synthesis front-ends supported by VTR

E. Enhancements to the VTR Flow

The VTR flow has traditionally been using Odin II [46] as its synthesis front-end. For the first version of the Koios suite [9], we found that some of the benchmark circuits could not be synthesized using Odin II as it only supports a subset of the Verilog-2005 standard. Some other benchmarks were also originally written in SystemVerilog which is not supported by Odin II. Therefore, we had to add support for some of the commonly used Verilog constructs to Odin II or re-write some of the benchmarks using only the supported subset of the Verilog-2005 standard. However, this was a very tedious and labor-intensive process that restricted the extension of the Koios suite to include more benchmarks that were written in SystemVerilog or used unsupported Verilog syntax. This initiated a new effort to improve the language coverage of the VTR synthesis front-end using a combination of the Yosys synthesis tool and Odin II [47].

Yosys is an open-source synthesis engine with extensive Verilog-2005 and SystemVerilog support [48]. Whereas most commercial synthesis tools are closed source, Yosys offers a flexible and open-interface synthesis process, which is valuable for developing new and customized synthesis algorithms. However, Yosys is totally agnostic to the target FPGA architecture and thus limits opportunities for architecture-aware logic inference (i.e. automatically inferring logic that can be mapped to hard blocks). Therefore, in this newly developed Yosys+Odin flow, Yosys provides better language coverage support and performs HDL elaboration followed by coarse-grained optimizations. After that, Odin II performs partial technology mapping based on the target FPGA architecture using a mix of genetic algorithms [49] and trade-off analysis of hard vs. soft logic inference [50], and then writes out the final netlist as illustrated in Fig. 3.

Our Koios 2.0 suite includes a variety of new benchmarks that use Verilog constructs that are not supported by Odin II, and therefore was used for developing and testing the new hybrid synthesis front-end. During this process, the Koios benchmarks helped identify and fix several issues such as:

- Unlike the Odin-only flow, the new hybrid front-end produced netlists with randomly-generated net names that cannot be traced back to their HDL declarations, which made debugging CAD flow errors significantly harder.

- Yosys used different names for the clock and reset signals of netlist atoms that can be packed into the same hard block (e.g. two multiplications to the same DSP block) which prevented VPR from packing them together even though they are actually connected to the same clock and reset signals, leading to higher resource utilization compared to the Odin-only flow.
- The new hybrid front-end generated a single black-box module definition for each of the hard block models specified in the VTR architecture description file. In some cases when the hard block has different modes of operation with different interface widths, this would result in technology mapping failures.

More recently, a modified version of Yosys, adapted from the Verilog-to-Bitstream tool [51], is also being integrated into the VTR flow to provide the option of performing all the synthesis steps solely using Yosys as shown in Fig. 3. However, this flow is not currently used for the Koios benchmarks.

The Koios benchmarks were also useful in exercising different parts of the tool flow that were not extensively tested before and identifying subtle bugs/issues in them. For example, multiple of the Koios circuits make heavy use of cascaded chains of multiple DSP blocks in a column to implement efficient dot product operations for DL. This presents additional placement constraints since the DSP blocks in a chain have to be initially placed and then moved around during placement optimizations as a single combined molecule to maintain the placement legality. Since VPR picks the device grid size based on the number of required blocks of each type without considering these additional placement constraints for cascaded DSPs, some of the Koios benchmarks (`tpu_like.small.os`, `dla_like.large` and `bwave_like.fixed.large`) were failing at the initial placement stage due to the absence of a legal solution given the predetermined device grid size. For example, a design can have a cascaded chain of N DSP blocks left to place and the device still has enough DSP blocks available but split across different columns (i.e. not in consecutive locations along a column). This results in a failure since no legal solution exists at this device grid size which was decided earlier in the flow based solely on the number of required blocks. A straightforward workaround is to manually specify a slightly bigger grid size for the failing benchmarks or a maximum resource utilization target for VPR when automatically sizing the device grid (the latter workaround is currently used for running the failing Koios benchmarks). It can also be fixed by iteratively increasing the grid size during initial placement in such cases until a legal solution is found. We flagged this issue and its suggested solution, which will be implemented by the VPR team in a future release.

F. Availability and Usage

The Koios 2.0 benchmarks are available at this link: `https://tinyurl.com/vtrkoios`. They have been tested and work out-of-the-box with the VTR flow. Scripts to automatically run and generate QoR for these benchmarks are also provided. In addition, we use the Titan flow [15] to generate the netlist (BLIF) files of the Koios benchmarks for the Stratix-IV FPGA architecture. These netlists can be used to directly run placement and routing using VPR without the need of an Intel Quartus license for running the synthesis front-end in the Titan flow. The BLIF files can be downloaded separately from this link: `https://tinyurl.com/koiosblif`.

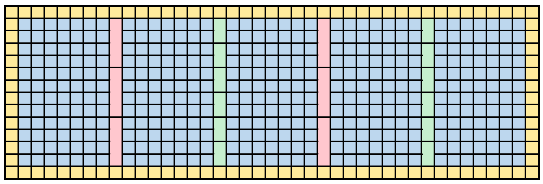


Fig. 4: FPGA architecture (not to scale) used for experimenting with Koios 2.0 benchmarks. Blue = Logic Block, Green = Block RAM, Red = DSP Slice, Yellow = Input Output Block

These benchmarks are implemented and curated in this suite to be used for FPGA architecture exploration and CAD tool optimization. They aim to accurately capture all the different circuit structures and compositions, but should not be expected to be deployed as standalone functional designs. We are confident that these circuits are structurally correct, and we have verified their high-level functionality to the best of our ability. However, full functional verification on many different test cases is beyond the scope of this work.

IV. Benchmark Results

A. Experimental Setup

We use the latest version of VTR 8 [8] for all our experiments in this paper. When running VTR, we provide an SDC (Synopsys Design Constraints) file in which the target clock period is set to 0 (i.e. VTR will optimize the design for maximum clock frequency). We also disable timing analysis for paths to/from the FPGA IOs. Unless stated otherwise, we run VTR with auto layout enabled (meaning the grid size expands based on the resources required by the design), the default timing-driven routing option with a maximum of 150 routing iterations, and a fixed channel width of 300 wires. All reported results are the average of three runs with different seeds. For experiments in which we report VTR flow runtime and peak memory usage, we use an Intel Xeon CPU E5-2430 running at 2.5 GHz with 64 GB of memory.

B. FPGA Architecture Used

We develop a new FPGA architecture description file to capture some relevant features of modern FPGAs. This architecture description file is also open sourced along with the benchmark suite. The delays and areas of all the FPGA blocks, including the DSP tiles, are obtained from COFFE [52] using a 22 nm technology node from PTM [53]. The circuits in this architecture are optimized for area-delay product, which leads to relatively higher delays compared to performance-optimized commercial FPGAs such as the Arria 10 family. Figure 4 shows a representation of this architecture, which is described in the rest of this subsection.

1) Floorplan: The FPGA contains columns of logic blocks, DSPs and BRAMs. Both DSP and BRAM columns repeat every 16 columns and are interleaved such that every 8th column is a DSP or a BRAM. The DSP and BRAM tiles are 4 and 2 rows high respectively, and the IO pads are arranged along the perimeter of the FPGA.

2) Routing Architecture: The architecture uses unidirectional routing with wire segments of length 4 (260 out of 300 wires) and length 16 (40 out of 300 wires). The length 16 wires do not directly connect to block pins and are only accessible from the length 4 wires. Switches appear after every 4 blocks on the length 16 wires. The switch blocks use a custom switching pattern based on the Stratix-IV-like architecture used in the Titan flow [15]. The input and output flexibility of connection blocks are set to 0.15 and 0.1, respectively.

3) Logic Blocks: Each logic block (LB) contains 10 basic logic elements (BLEs) similar to that in the Intel Stratix-10-like architecture from [20]. Each block has 60 input pins, 40 output pins, and a 50% sparsely populated local input crossbar. Each BLE has a 6-input LUT which can be fractured into two 5-input LUTs. The BLE also has 2 flip-flops and 2 bits of arithmetic with dedicated carry chains between LBs. Each BLE has 8 inputs and 4 optionally registered outputs.

4) DSP Slices: This architecture has a complex DSP block that supports most of the operating modes in the state-of-the-art Intel Agilex DSP block [54]. Multiple fixed point (9x9, 18x19, 27x27) and floating point (IEEE 32-bit (`fp32`), IEEE 16-bit (`fp16`) and Brain floating point (`bfloat16`)) precisions are supported. In addition, the DSP block has dedicated output chains for cascading several DSP blocks in the same column for efficient dot product structures.

5) BRAMs: BRAM blocks have a capacity of 20 Kb and have registered inputs and outputs. True and simple dual port modes are supported. In the simple dual port mode, a BRAM can be configured as: 512×40b, 1024×20b and 2048×10b, while the widest mode is not supported in true dual port mode. The delays and areas of a BRAM block are obtained by interpolation between the values obtained from COFFE for 16 Kb and 32 Kb BRAMs.

Some benchmarks in Koios use advanced DSP features that are available in the FPGA architecture described above. This is done by instantiating DSP hard macros directly into the RTL when implementing natively-supported `fp16` multiplications or DSP cascaded structures for example. Similarly, BRAMs are also instantiated as hard macros in the RTL. Although these hard macros are architecture-specific, users can still use the Koios benchmarks with other FPGA architectures by replacing these RTL instantiations with their alternatives. To improve the usability of the Koios benchmarks, we also implement the same functionality of the architecture-specific hard macros using behavioral Verilog, and allow users to switch between the hard macro and behavioral implementations using pre-processor directives (i.e. `ifdefs`). By disabling the `complex_dsp` and `hard_mem` directives, the benchmarks become completely architecture-agnostic and can be used with any FPGA architecture description file. In this case, the synthesis tool infers the hard blocks to be used and generates a netlist containing hard macro instances available in the user's FPGA architecture. If no hard blocks are available in the FPGA, the code will just be mapped to FPGA soft logic. We have verified running the benchmarks without these directives for the FPGA architecture described in this section and the VTR flagship architecture.

This makes the Koios benchmarks also suitable for evaluating the addition of new hard blocks to an FPGA architecture, similar to some recent DL-optimized FPGAs [3], [23]. To perform such studies, users can either: (1) modify the synthesis engine to automatically extract specific patterns from the Verilog designs and map them to the new blocks, or (2) modify the benchmarks to instantiate these new blocks as hard macros (defined in the VTR architecture file).

C. Results of the Koios 2.0 Benchmarks

Table IV shows the VTR results for the Koios 2.0 benchmarks when running them with the FPGA architecture described in Section IV-B. The results show that these designs, with sizes ranging from 12K to 1.6M netlist primitives, are deeply pipelined with 27 out of the 40 benchmarks having critical paths with 6 or less logic levels on them. The benchmarks are also highly diverse in heterogeneity, with varying circuit compositions between soft logic, DSPs, and BRAMs. For example, some designs do not utilize any BRAMs since they either implement only the workload datapath (e.g. `gemm_layer` and `softmax`) or use distributed registers for storage (e.g. `bnn`). On the other hand, there are other BRAM-intensive designs such as `tdarknet_like.large` with $4,400$ BRAMs utilized. Similarly, with DSPs, there are some designs that use very few or no DSPs (e.g. `conv_layer_hls` and `reduction_layer`) as they mostly implement other non-multiplication operations in DL workloads such as pop-count or max/min/add reduction. Other designs are DSP-intensive (e.g. `deepfreeze.style2`) with over $1,700$ DSP blocks. Table IV also shows that different types of resources are the grid size limiting factor for different benchmarks in our suite. The majority of the designs are bound by hard blocks, as indicated by the bold entries in the table, which emphasizes that these benchmarks can be useful for exploring new DSP and BRAM architectures.

Most of the designs in the Koios 2.0 suite can achieve reasonably high operating frequencies up to 249 MHz and an average of 124 MHz. The FPGA architecture used for our experiments is not very fast. The delays in the architecture are based on area-delay-optimized PTM models (with raw delays similar to 40 nm Stratix-IV). Changing the delays of FPGA resources to those typical of a high-speed ($\leq$14 nm) device would increase the frequency by $>2\times$. The `lenet` design is a clear outlier with a frequency of 53.9 MHz. This design is generated by HLS and has a very high logic depth of 34. The total routed wirelength of the benchmarks are largely correlated with the circuit size and ranges from 102K up to 11.4M units of length 1 wire segments.

The top graph in Fig. 5 plots the VTR flow runtime for each benchmark. The trendline shows that the runtime grows almost

TABLE IV: VTR results of the Koios 2.0 benchmarks

Benchmark	Netlist Primitives	Logic Depth	Used IOs	Used LBs	Used DSPs	Used BRAMs	Max. Freq.	Routed Wirelength	Elapsed Time	Peak Memory
dla_like.large	1608867	5	819	28201	**1376**	864	107.4	11445	1140.0	15733.5
clstm_like.large	1083855	3	1518	26341	**961**	739	105.6	5785	842.7	12901.0
deepfreeze.style3	759656	3	540	18499	340	**3489**	116.3	5380	289.3	16131.7
clstm_like.medium	743071	3	1230	17854	**661**	498	113.9	3767	400.4	8805.7
deepfreeze.style1	687669	3	540	15115	700	**1999**	135.2	4673	243.0	10172.1
dla_like.medium	600492	5	411	10656	**400**	312	140.6	2920	209.0	5408.4
deepfreeze.style2	470421	3	540	12896	**1762**	1387	62.6	3466	246.3	15574.3
proxy.2	439725	8	574	8921	330	**1099**	130.9	3293	228.9	5796.2
clstm_like.small	402331	3	942	9396	**361**	257	131.3	1821	100.8	4739.3
tdarknet_like.large	391291	5	46	13574	367	**4400**	72.7	4173	775.4	18456.7
proxy.4	391195	7	2392	7768	**757**	1189	101.3	4510	401.9	7439.1
proxy.1	358143	7	1113	5989	**1037**	619	125.3	4325	206.8	9503.0
bwave_like.float.large	310527	6	1093	9699	**640**	1182	93.9	4440	114.5	6522.4
proxy.3	304125	10	1036	9585	107	**847**	96.8	2491	124.7	4569.3
dla_like.small	260199	5	207	4799	**128**	132	160.7	998	59.3	2143.0
proxy.7	248950	7	498	4937	**302**	492	114.2	2167	135.9	3214.4
lstm	247060	7	2677	5060	**610**	305	121.8	2129	272.2	5767.3
proxy.6	206539	3	1025	3403	**300**	406	134.7	1720	174.3	3053.7
bnn	204601	3	382	**5694**	63	0	131.0	1184	17.2	2171.0
lenet	190809	34	140	7417	**497**	820	53.9	3250	671.4	5850.0
dnnweaver	189706	6	3531	5552	**288**	**1139**	82.4	2921	49.7	5258.4
tdarknet_like.small	157431	6	46	6974	**90**	**3978**	63.8	2657	217.4	16043.7
proxy.8	150264	7	1002	3047	**367**	378	110.9	1266	67.2	3325.1
proxy.5	147618	7	785	3199	**283**	236	108.1	1227	70.2	2768.5
bwave_like.float.small	84893	6	200	2625	144	**358**	129.1	936	14.2	1802.7
tpu_like.large.ws	78335	8	1190	3011	**1066**	116	100.2	961	87.9	8848.8
tpu_like.large.os	70946	5	1188	1596	**1064**	64	120.4	2028	95.4	8826.8
gemm_layer	64765	4	1779	2001	**200**	0	173.9	789	17.6	1897.6
bwave_like.fixed.large	54871	6	328	1299	**562**	511	104.2	1816	32.4	5938.8
attention_layer	54865	7	1089	1455	**137**	194	124.5	480	18.6	1328.6
conv_layer	37268	4	156	938	**42**	56	218.6	245	6.5	562.0
robot_rl	30529	6	387	1285	18	**96**	148.8	232	6.0	522.9
tpu_like.small.ws	27097	7	646	1034	**278**	58	118.8	288	15.1	2407.2
tpu_like.small.os	21962	5	644	538	**276**	32	156.7	416	13.9	2381.5
reduction_layer	18323	6	54	805	0	**52**	147.4	183	1.9	340.2
spmv	17734	6	99	503	32	**232**	178.4	221	4.0	946.1
bwave_like.fixed.small	16632	5	198	404	**139**	170	132.7	397	5.2	1293.1
eltwise_layer	16187	4	249	355	**50**	72	249.1	193	2.6	472.8
softmax	13177	10	552	512	**53**	0	114.6	126	2.3	492.1
conv_layer_hls	12097	3	**3299**	1717	12	21	151.1	102	12.2	3983.8

Frequency is in MHz, Routed Wirelength is 1000 length-1 segments, Elapsed Time is in minutes, and Peak Memory is in MBs.

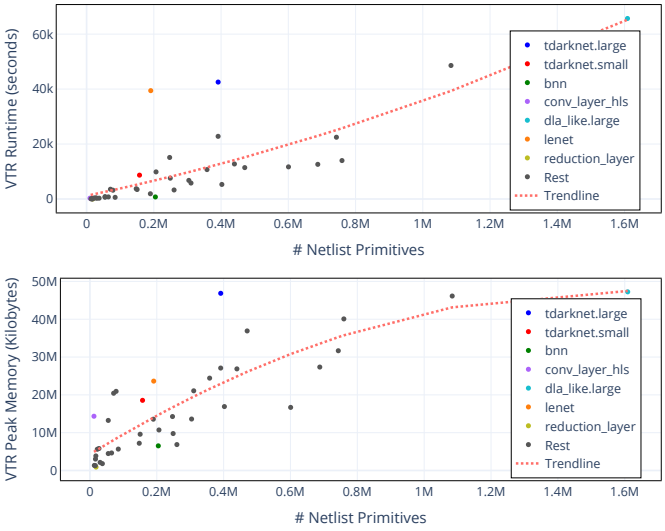
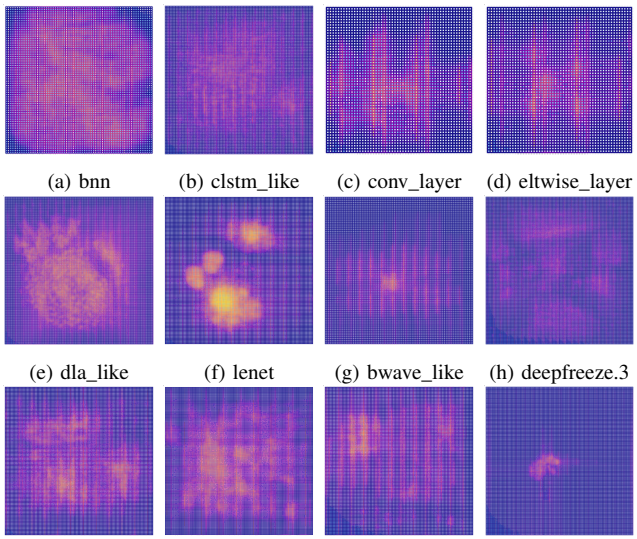


Fig. 5: VTR runtime (top) and peak memory usage (bottom) for the Koios benchmarks

linearly with the number of netlist primitives in the circuits. There are some notable exceptions; `lenet` and `tdarknet` designs have very high runtime for their number of netlist primitives. Looking at the components of runtime, we see that in most benchmarks, ABC (the tool that performs logic optimization and techmapping in the VTR flow) takes more time compared to Odin/Yosys and VPR. The bottom graph in Fig. 5 plots the VTR flow peak memory usage for the Koios benchmarks. The trendline shows a sub-linear growth in peak memory requirement as the number of netlist primitives increases. The `lenet` and `tdarknet` designs again have very high memory usage for their size, and we observe that VPR consumes the majority of used memory compared to Odin/Yosys and ABC.

The routing heat maps for some Koios benchmarks are shown in Fig. 6, where the lighter color correspond to higher routing congestion. The routing heat maps look very different for different designs, highlighting the diversity in routing requirements and patterns of the benchmarks, which exercises the placement and routing algorithms in different ways. Some benchmarks have a very regular pattern (e.g. `bnn`), which implies heavy usage of LBs (soft logic). In other benchmarks, we see high routing congestion along columns of hard blocks (e.g. `dnnweaver`).



(a) bnn (b) clstm_like (c) conv_layer (d) eltwise_layer

(e) dla_like (f) lenet (g) bwave_like (h) deepfreeze.3

(i) proxy.2 (j) proxy.7 (k) dnnweaver (l) tpu.ws

Fig. 6: Routing utilization heatmaps for some Koios benchmarks

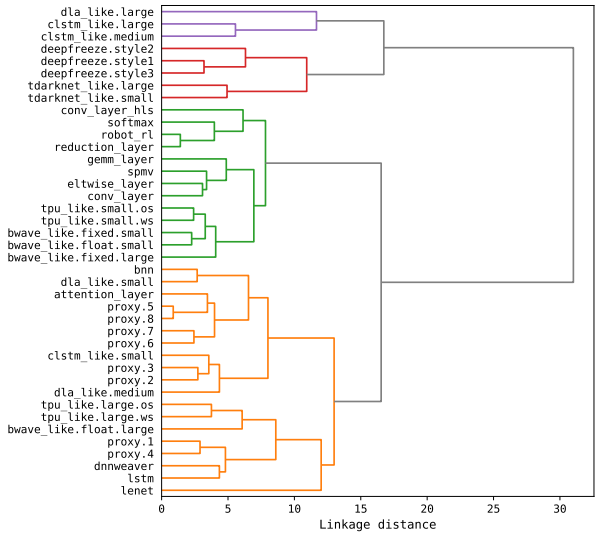


Fig. 7: Dendrogram showing similarity between Koios 2.0 benchmarks

D. Statistical Analysis

To perform statistical analysis on the Koios 2.0 suite, we collect a large number of metrics (35 metrics) for all benchmarks. Some metrics other than those in Table IV are logic depth, max non-global fanout, average wire segments per net, max routing channel utilization, number of near critical connections, number of blocks before and after clustering, and the maximum number of wire segments used by a net. However, it is difficult to manually investigate the data and conduct meaningful analysis. Hence, we perform principal component analysis (PCA) [55], [56] on the collected data, which converts N variables into a smaller group of m linearly uncorrelated variables known as the *principal components (PCs)*. In our analysis, we use $m = 4$. Each PC is a linear combination of different features or variables with a certain weight. The first PC covers the majority of the variance, and subsequent PCs cover diminishing variances. By eliminating components with lower variance values, the dimensionality of the data set can be reduced. We also examine benchmark similarity by hierarchically clustering them. The Euclidean distance of various metrics (or variables) is used to calculate how similar two benchmarks are. The output of this clustering can be displayed as a tree or dendrogram in which smaller linkage distance between two benchmarks indicates higher similarity between their metrics.

Fig. 7 shows the dendrogram plot for the Koios benchmarks. The x-axis shows the linkage distance between the different benchmarks on the y-axis. The absolute value of the distance does not matter, but the relative value between benchmarks does, and the ordering of benchmarks on the y-axis has no special significance. We can observe that there is no one benchmark that is particularly unique. If we draw a vertical line at linkage distance = 15 (for example), we can then divide the benchmark suite into the 4 subsets shown in different colors. A user with limited compute resources or in early stage experiments can choose one or a few benchmarks from each subset to get the maximum coverage for their experiments. As another example, this analysis shows that among the 8 proxy benchmarks, {1,4}, {2,3} and {5,6,7,8} are three groups that have very similar characteristics across the circuits in each of them. This means that a user could choose one proxy benchmark from each group as a representative benchmark in case of limited resources/time. It also means that we can use the proxy generator to design unique proxy benchmarks that have higher linkage distances in the future.

Fig. 8 shows a scatter plot of all the Koios benchmarks based on the first two PCs covering 65% of the variance (50% in PC1 and 15% in PC2). The PCA analysis provides coefficients for each of the metrics to identify the main contributors to each PC. PC1 is mainly dominated by metrics related to the size (netlist primitives, CLB usage, routing wirelength, runtime). PC2 is dominated by average net length, near critical connections, device size and frequency. The 4 benchmarks at the extreme opposites of PC1 and PC2 in Fig. 8 are `tdarknet_like.small`, `eltwise_layer`, `bnn` and `dla_like.large`. These 4 benchmarks belong to different groups from the dendrogram in Fig. 7. We also see that the proxy benchmarks appear towards the center of the scatter plots, implying that they represent the common benchmarks of the suite. Thus, if a user is constrained on resources, a representative subset of the benchmark suite could be the 4 extreme benchmarks and one or more of the proxy benchmarks for example. The next two PCs cover a small amount of variability (7.5% in PC3 and 6.5% in PC4). PC3 is dominated by the logic depth and maximum routing channel utilization. PC4 is dominated by maximum non-global fanout and maximum net length.

V. Comparison to Other Benchmark Suites

A. Methodology

In this section, we compare various properties of our Koios benchmarks to those of other existing non-DL-targeted benchmarks that are commonly used to drive FPGA architecture and CAD research. The most relevant suite for

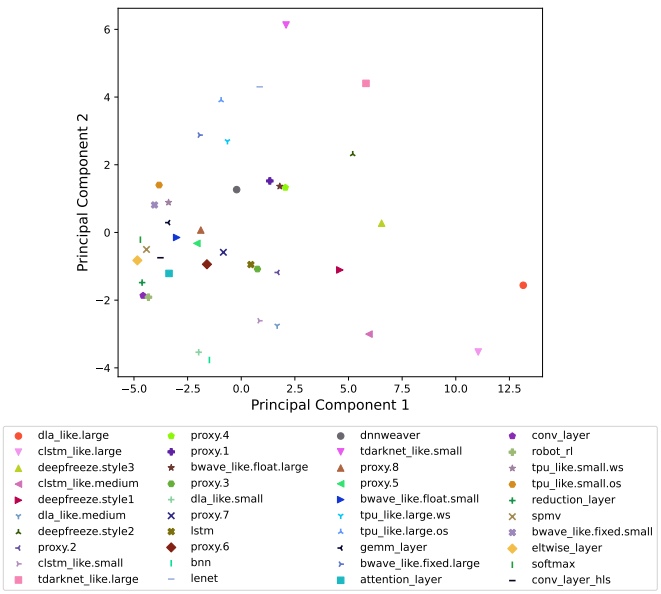


Fig. 8: Analyzing the Koios 2.0 benchmarks using PCA

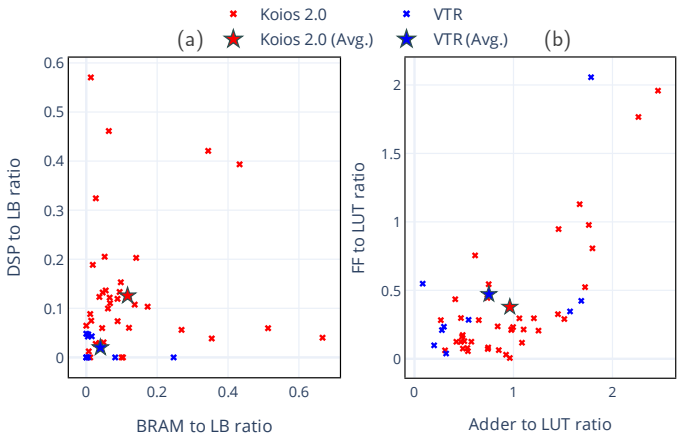


Fig. 9: Comparing circuit compositions of Koios & VTR benchmarks: (a) DSP/BRAM to LB and (b) FF/adder to LUT ratios

comparison is the VTR benchmark suite, because these are compatible with the same fully open source VTR flow. Other existing suites are either too small and do not represent realistic modern use cases of FPGAs or depend partially on commercial CAD tools. For this comparison, we only use the VTR benchmarks with more than $10,000$ netlist primitives (9 benchmarks), which is a common practice in CAD-related studies [57]. Smaller designs are not representative of realistic benchmarks, and they cannot be used to derive any reliable conclusions. We use the same VTR settings and architecture file as in Section IV.

In addition, we also compare the Koios benchmarks to the Titan23 benchmarks [15]. The Titan benchmarks are not compatible with the fully open source VTR flow and depend on the Intel Quartus tool to perform logic synthesis and generate netlist BLIF files. Therefore, they can only be placed and routed using the Stratix-IV-like architecture capture in VTR, which limits their usability for FPGA architecture studies. However, they are commonly used as large representative benchmarks for FPGA CAD research and for evaluating QoR of different CAD algorithms/flows. For this comparison, we run the BLIF netlists of the Titan benchmarks provided in the v1.3.1 release of Titan through VPR and the Koios HDL benchmarks through the end-to-end VTR flow. We use the same VPR settings from the official VTR Titan regression tests for running both sets of benchmarks.

Finally, we also present a QoR comparison between VPR and Quartus using the Koios benchmarks implemented on the Stratix-IV FPGA architecture, since this is currently the only Intel FPGA architecture with a corresponding architecture capture in VTR. In this experiment, we run Intel Quartus Prime 20.1 using the default compiler effort settings (i.e. `STANDARD_FIT` mode). For a fair comparison to VPR with auto layout, we set the Stratix-IV device in Quartus to `AUTO` which automatically selects the smallest Stratix-IV device that can fit the given design. On the other hand, we synthesize the Koios benchmarks for Stratix IV using Intel Quartus Prime 20.1 and then use the `vqm2blif` tool from the Titan flow to generate Koios BLIF netlists. Then, we run them through VPR with the same settings used for evaluating QoR in [8]. We set the placement `inner_num` to 1.0, the router `astar_fac` to 1.0, and the router iterations to 400. Both Quartus and VPR were also given equivalent timing constraints with an aggressive 1ns clock period target and paths to/from external IOs constrained on a virtual IO clock as in [8].

B. Comparison to the VTR Benchmarks

Fig. 9a shows a scatter plot of the DSP and BRAM to LB ratios for both the Koios (red) and VTR (blue) benchmarks as metrics for their DSP and memory density. The individual ratios for each of the benchmarks are shown by $(\times)$ symbols while the average across the whole benchmark suite is marked by the stars. The figure shows that, on average, the Koios benchmarks are more DSP and memory rich than the VTR benchmarks; it has $2.9\times$ and $6.2\times$ higher DSP to LB and BRAM to LB ratios, respectively. The individual benchmarks of the Koios suite are also more scattered and varying across the spectrum of DSP and BRAM compositions. More importantly, it shows that most of the VTR benchmarks have very low DSP and BRAM densities (except for the only `stereovision2` outlier circuit), making them inadequate for evaluating any DSP or BRAM architecture modifications.

Fig. 9b has a similar plot for FF and single-bit adder to LUT ratios. It shows that the Koios suite has $1.28\times$ higher ratio between FFs and LUTs which reflects their deeply pipelined nature, and 20% lower adder to LUT ratio compared to the VTR suite. However, the average adder to LUT ratio of the VTR suite is significantly skewed by a single benchmark (`stereovision2`) which has $60,753$ 1-bit adders and only $29,541$ LUTs. If we exclude this outlier, the Koios benchmarks have a $1.4\times$ higher average adder to LUT ratio.

Fig. 10 illustrates averages and ranges of key metrics for both the Koios and VTR benchmark suites. Fig. 10a-d show that the Koios benchmarks have $4.5\times$ more netlist primitives, $4.9\times$ larger non-global fanouts, $2.07\times$ more near (top 10%) critical connections, and $1.5\times$ higher frequencies on average compared to the VTR benchmarks. The Koios benchmarks are

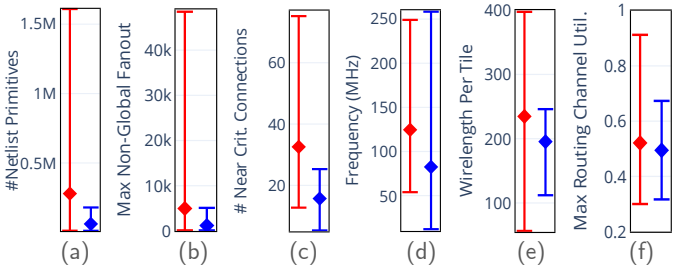


Fig. 10: Averages and ranges of key metrics of Koios 2.0 (Red) & VTR (Blue) suites.

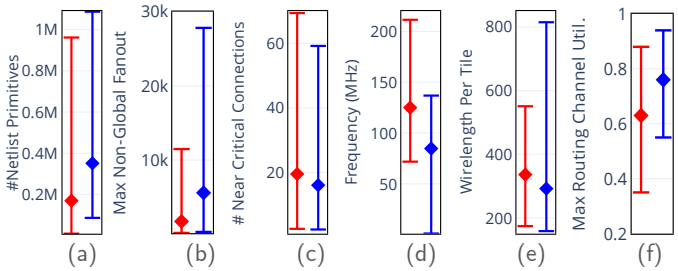


Fig. 11: Averages and ranges of key metrics of Koios 2.0 (Red) & Titan (Blue) suites.

also scattered across a much wider range of values for each of those metrics. Fig. 10e shows that Koios circuits have 20% higher average routed wirelength per tile compared to VTR benchmarks. Fig. 10f shows that the circuit with the highest max routing utilization in Koios has 35% higher utilization compared to the circuit with max routing utilization in VTR benchmarks. Koios designs also have an average of 6 logic levels on the critical path, compared to 30 levels for the VTR benchmarks. This reflects the deeply pipelined nature of these benchmarks, which is a key property of modern FPGA designs.

C. Comparison to the Titan Benchmarks

Only 22 out of 23 Titan benchmarks were successfully placed and routed. The largest circuit (`gaussianblur`) failed with runtime exceeding 4 days, and therefore was excluded from the comparison. On the other hand, some of the Koios benchmarks consume more resources than that available in the largest Stratix-IV device, and thus were excluded for a fair comparison (since the Titan benchmarks had to be synthesized through Quartus to a real Stratix-IV device). 6 out of the 40 Koios circuits were excluded for DSP/BRAM limitations and another 11 were excluded for IO limitations, leaving 23 Koios benchmarks valid for this comparison.

Both Titan and Koios 2.0 suites are heterogeneous - they have a large number of DSPs and BRAMs. On average, the 23 Koios designs have 2.17× DSPs, 0.66× memory bits and 0.51× routed wirelength, compared to the 22 Titan designs. Fig. 11a shows that the Koios benchmarks are smaller; there are 2.06× more netlist primitives on average in Titan benchmarks. Koios benchmarks have 3× lower max non-global fanout (Fig. 11b) than Titan benchmarks. However, Koios benchmarks have higher min, max and average number of near-critical connections, compared to Titan benchmarks, as seen in Fig. 11c. Koios benchmarks run at significantly faster frequency (Fig. 11d) compared to Titan benchmarks. The fastest Koios 2.0 design runs at 1.55× higher frequency compared to the fastest Titan design. There are designs with very low frequency (minimum=1.1MHz) in the Titan suite. The average wirelength per tile (Fig. 11e) is 15% higher in Koios benchmarks. Fig. 11f shows that Koios benchmarks have a much wider range of maximum routing utilization, compared to Titan benchmarks, although the max routing utilization is high on average in Titan benchmarks. Overall, both suites pose challenging problems to CAD flows, but there are some peculiar characteristics of Koios 2.0 benchmarks, like high frequency and heavy DSP usage, owing to them belonging to the DL domain.

D. QoR Comparison of VPR and Quartus

Table V presents the detailed QoR comparison of VPR and Quartus for a subset of the Koios benchmarks that could fit on Stratix-IV devices. For most of the benchmarks, VPR packed denser logic clusters, resulting in 33% less Logic Blocks (column 'LB') on average. Although denser logic clustering in VPR was previously reported in [15] and later reduced in [8] resulting in better critical path delays, the Koios benchmarks show a much bigger difference in logic packing density between VPR and Quartus compared to the 5% difference in [8] which uses an older version of Quartus. VPR also uses 42% more DSP Slices (column 'DSP') than Quartus, with some benchmarks (e.g. `bwave_like.fixed.large`) using up to 4×. The reason is that VPR, due to its generality, cannot efficiently map multiplication primitives to DSP blocks in its complex modes of operation. On the other hand, Quartus searches for specific patterns in the circuit netlists that can be efficiently mapped to the target device DSP blocks. For BRAMs, the results show that VPR rarely makes use of the bigger 144 Kb BRAMs (column 'M144K') which it uses in only one benchmark (`lenet`). In contrast, Quartus uses these bigger BRAMs in 7 other circuits, as indicated by the BRAM counts in brackets in Table V. For these benchmarks, VPR maps all logical memories to the smaller 9 Kb BRAMs (column 'M9K') resulting in a 1.9× higher utilization of these blocks when averaged across the 7 benchmarks, which translates to a 24% increase across the whole suite. VPR also results in 1.46× higher total routed wirelength (column 'WL') and 1.36× longer critical path delays (column 'CP') compared to Quartus. These gaps are higher than the 1.26× higher total routed wirelength and 1.2× longer critical path delay reported in [8] on the less heterogeneous and less DSP-intensive Titan benchmarks. These bigger gaps can be attributed to the less efficient packing and mapping of hard blocks discussed above which are more heavily used in the Koios benchmarks, and also their deeply pipelined nature. This highlights the value of having more challenging benchmarks that can exercise the CAD tools in different ways.

VI. Case Studies

Koios benchmarks are architecture-agnostic and do not depend on commercial tools for any portion of the FPGA CAD flow. Thus, they can be used to perform flexible FPGA

TABLE V: VPR and Quartus QoR comparison on Koios. Numbers are ratios of VPR:Quartus results, '–' represents unutilized resource for both, and numbers in brackets are the absolute count of resources used by Quartus when VPR used none.

Benchmark	LB	DSP	M9K	M144K	WL	CP
attention_layer	0.67	0.69	1.07	–	0.73	1.03
bnn	0.84	–	–	–	0.54	1.33
bwave_like.fixed.large	0.86	4.00	1.00	–	2.23	1.58
bwave_like.fixed.small	0.37	4.00	1.00	–	2.18	1.59
bwave_like.float.large	0.76	3.00	1.00	–	1.23	1.02
bwave_like.float.small	0.53	2.96	1.00	–	2.62	1.29
conv_layer	0.53	0.89	1.00	–	1.41	1.7
dla_like.large	1.15	1.73	1.27	(24)	1.35	1.25
dla_like.medium	1.10	1.67	1.42	(12)	2.04	1.14
dla_like.small	0.95	0.76	1.58	(6)	1.54	1.04
eltwise_layer	0.57	0.50	1.00	–	1.30	1.36
lenet	0.80	0.85	0.77	1.00	1.37	1.55
proxy.1	0.89	3.00	1.00	–	1.06	1.14
proxy.2	0.68	1.23	1.87	(60)	2.52	2.24
proxy.3	0.52	0.92	2.10	(60)	1.54	1.15
proxy.5	0.46	0.93	2.99	(16)	1.33	1.64
proxy.7	0.66	0.88	2.77	(48)	1.36	1.44
reduction_layer	0.76	–	1.00	–	0.97	1.44
robot_rl	0.78	2.00	1.00	–	1.35	1.15
softmax	0.68	0.63	–	–	1.01	1.17
spmv	0.62	1.00	1.00	–	1.54	1.23
tpu_like.small.os	0.35	2.79	1.00	–	2.87	1.88
tpu_like.small.ws	0.50	1.89	1.00	–	2.16	1.55
Geomean	**0.67**	**1.42**	**1.24**		**1.46**	**1.36**

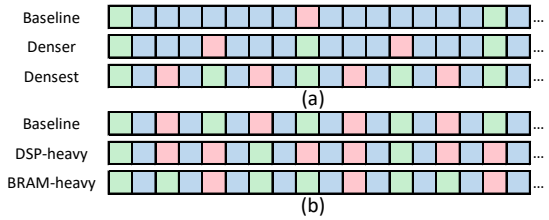


Fig. 12: FPGA layouts for the architectures used in our case studies. Blue = Logic Block, Green = Block RAM, Red = DSP Slice

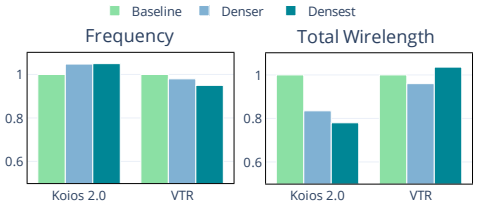


Fig. 13: Effect of varying the density of DSPs and BRAMs on Koios 2.0 and VTR benchmark suites

architecture and CAD exploration using the fully-open-source VTR flow. In this section, we present three example case studies to demonstrate that.

A. Case Study 1: Hard Blocks to Soft Logic Ratio

As shown in Table IV, our DL-focused circuits are highly heterogeneous (i.e. DSP and BRAM intensive). Thus, in our first case study, we vary the density of these hard blocks with respect to soft logic. We experiment with 3 different density levels, as shown in Fig. 12a, with 1:7, 1:3, and 1:1 ratio between hard block and soft logic columns for the baseline, denser, and densest architecture variations, respectively. We evaluate all three architecture variations using both Koios and VTR benchmarks. Fig. 13 shows the geomean frequency and total routed wirelength for both suites. For the DL-oriented Koios benchmarks, the frequency increases and wirelength decreases as the density of hard blocks increases. Since these benchmarks heavily utilize these blocks, increasing their density in the FPGA grid brings them closer to each other, which in turn reduces the critical path delays and total length of used wires. The densest architecture variation results in a 5.2% increase in frequency and 22% reduction in total wirelength on average across all benchmarks in the Koios suite. For the VTR benchmarks, wirelength is slightly improved for the denser variation (4% lower), before getting worse for the densest architecture. The frequency degrades for both denser and densest architectures. These results show that a higher density of DSPs and BRAMs is favorable for building DL-optimized FPGAs, at the cost of a slight or no degradation in QoR for the general VTR benchmarks (in the densest and denser architecture variations respectively).

TABLE VI: Effect of varying the FPGA's DSP to BRAM ratio

Metric	Arch.	Geo-mean	DSP-heavy tpu_like(L)	BRAM-heavy tdarknet_like(L)
Freq.	**Baseline**	125.6	102.5	86.2
	DSP-heavy	124.9	**110.4**	92.0
	BRAM-heavy	126.2	106.0	**114.6**
WL	**Baseline**	1065K	749K	3105K
	DSP-heavy	1065K	**720K**	3343K
	BRAM-heavy	1098K	781K	**3106K**
Grid	**Baseline**	109×109	224×224	190×190
	DSP-heavy	110×110	**210×210**	232×232
	BRAM-heavy	111×111	228×228	**167×167**

Frequency is in MHz, Wirelength (WL) is in units of length 1 wires.

B. Case Study 2: DSP to BRAM Ratio

In our first case study, we varied the ratio of hard blocks to soft logic while keeping a fixed 1:1 DSP to BRAM ratio. For the second case study, we carry over the best architecture variation for DL benchmarks from the first case study (i.e. densest). However, we vary the DSP to BRAM ratio between 2:1 and 1:2 to create DSP-heavy and BRAM-heavy variations respectively (in addition to the baseline with 1:1 ratio), as shown in Fig. 12b. Table VI presents the results of this experiment. It shows the geomean frequency, routed wirelength, and FPGA grid size for the whole Koios suite, as well as the results for a DSP-intensive benchmark (medium `tpu_like.large.ws`) and a BRAM-intensive benchmark (`tdarknet_like.large`). The geomean results do not show a strong trend that clearly favors a specific architecture. However, we observe that the DSP-heavy `tpu_like.large.ws` design has 7.7% higher frequency and 4% lower wirelength when implemented on the DSP-heavy architecture compared to the baseline. Similarly, the BRAM-heavy `tdarknet_like.large` benchmark has 33% higher frequency and requires a 23% smaller device when implemented on the BRAM-heavy architecture compared to the baseline. This highlights that Koios strikes a good balance between different circuit compositions and can be reliably used for DL-optimized FPGA architecture exploration.

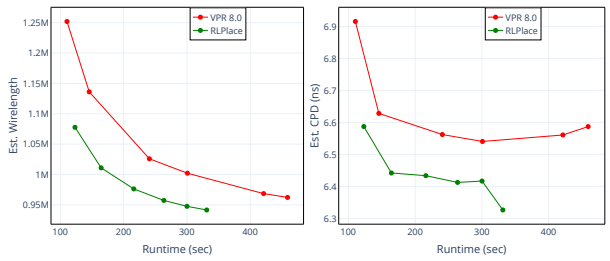


Fig. 14: Comparison of RLPlace and VPR8 on different runtime points for the Koios benchmarks

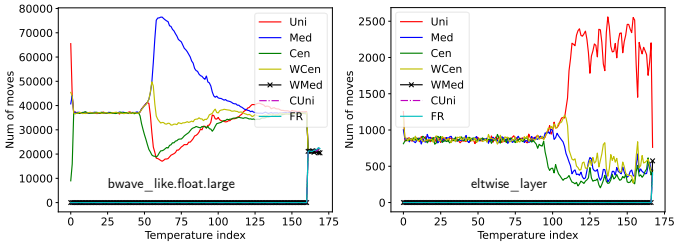


Fig. 15: Number of every move type chosen at each annealing temperature for different designs

C. Case Study 3: RLPlace Behavior

RLPlace [58] is a simulated-annealing-based FPGA placer built on top of VPR8 [8] infrastructure. RLPlace proposes the use of multiple smart directed perturbations and a reinforcement learning (RL) agent to balance between the different move types and pick the most efficient one based on the architecture, design, and annealing stage. RLPlace results in 2× faster placement on the VTR and Titan23 benchmarks [58]. In this case study, we use the Koios benchmarks to compare the performance of the baseline VPR 8 placer [8] and RLPlace [58]. As CPU runtime and the QoR are both important metrics, we compare the quality/runtime tradeoff curve for both tools by varying the number of moves per annealing temperature. Fig. 14 shows the quality/runtime tradeoff for the post-placement estimated wirelength (WL) and critical path delay (CPD) obtained on all the Koios benchmarks. To avoid CAD noise, the results of each design are averaged first over three different seeds. Then, all the results are geometrically averaged to get a single data point on the curve.

The results show that RLPlace is outperforming the VPR 8 placer in all the tradeoff points. This means that RLPlace can get the same or better QoR in 50% of the runtime and it can achieve better QoR for both timing and wirelength at the same runtime budget. These results match (and outperform for CPD) what was reported by RLPlace on the VTR and Titan benchmarks, giving more credibility to the gains of RLPlace. Fig. 15 shows the move type distribution (i.e. the number of times each move type was chosen by the RL agent) at each annealing temperature for 2 Koios designs. Compared to the patterns of the RL agent's choices reported by RLPlace in [58], some of the designs in Koios have totally different and unique patterns. For example, Fig. 15 shows how the agent prefers the Median move over all the other move types most of the annealing time for the `bwave_like.float.large` design (which was not the case for any VTR/Titan designs). Another example shown is the `eltwise_layer` benchmark, in which the RL agent prefers the random move starting from the middle of the anneal to its end. While these results show that the RL agent can learn to automatically adjust its online decisions to maximize QoR/performance, they also highlight the importance of using a variety of benchmarks to evaluate new CAD algorithms.

VII. Conclusion

In this paper, we present Koios 2.0, a DL-focused benchmark suite for FPGA architecture and CAD research. This suite is a diverse collection of 40 curated benchmarks covering various facets of the DL acceleration landscape. The Koios 2.0 suite is open-sourced as a part of VTR, and we highly encourage the FPGA community to contribute to this benchmark suite to help build a better and bigger set of DL benchmarks that can guide the design of future FPGA architectures and CAD algorithms.

Acknowledgement

We are thankful to Alireza Azadi of the University of New Brunswick, and Sara Mahmoudi and Kimia Talaei from the University of Toronto for their help with debugging issues in CAD tools (ODIN and VPR). We would like to thank Helen Dai and Zach Zheng from the University of Toronto, and Aatman Borda, Daniel Rauch, Aishwarya Rajen, Samidh Mehta and Pragnesh Patel from the University of Texas at Austin for contributing to the benchmarks. We are grateful to the National Science Foundation (grant number 1763848), the Vector Institute for AI, and the Intel/VMWare Crossroads 3D FPGA Academic Research Center for funding support.

References

[1] M. Hall and V. Betz, "HPIPE: Heterogeneous Layer-Pipelined and Sparse-Aware CNN Inference for FPGAs," *arXiv preprint arXiv:2007.10451*, 2020.

[2] A. Boutros *et al.*, "Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs," in *International Conference on Field Programmable Technology (FPT)*, 2020.

[3] M. Langhammer *et al.*, "Stratix 10 NX Architecture and Applications," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021.

[4] E. Nurvitadhi *et al.*, "Why Compete When You Can Work Together: FPGA-ASIC Integration for Persistent RNNs," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.

[5] S. Ahmad *et al.*, "Xilinx First 7nm Device: Versal AI Core (VC1902)," in *Hot Chips Symposium*, 2019.

[6] J. Zhang *et al.*, "Frequency Improvement of Systolic Array-Based CNNs on FPGAs," in *International Symposium on Circuits and Systems (ISCAS)*, 2019.

[7] A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 4–29, 2021.

[8] K. E. Murray *et al.*, "VTR 8: High Performance CAD and Customizable FPGA Architecture Modelling," *ACM Transactions on Reconfigurable Technology Systems (TRETS)*, vol. 13, no. 2, 2020.

[9] A. Arora *et al.*, "Koios: A Deep Learning Benchmark Suite for FPGA Architecture and CAD Research," in *International Conference on Field-Programmable Logic and Applications (FPL)*, 2021.

[10] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," 1991.

[11] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," in *International Conference on Field-Programmable Logic and Applications (FPL)*, 1997.

[12] J. Allen. (2006) UMass RCG HDL Benchmark Collection. [Online]. Available: http://www.ecs.umass.edu/ece/tessier/rcg/benchmarks/

[13] P. Jamieson *et al.*, "Benchmarking and Evaluating Reconfigurable Architectures Targeting the Mobile Domain," *ACM Transactions on Design Automation of Electronic Systems*, vol. 15, no. 2, 2010.

[14] D. Chang *et al.*, "ERCBench: An Open-Source Benchmark Suite for Embedded and Reconfigurable Computing," *International Conference on Field Programmable Logic and Applications (FPL)*, 2010.

[15] K. E. Murray *et al.*, "Timing-Driven Titan: Enabling Large Benchmarks and Exploring the Gap between Academic and Commercial CAD," *ACM Transactions on Reconfigurable Technology Systems*, vol. 8, no. 2, 2015.

[16] "Berkeley logic interchange format (blif)," 1996.

[17] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez, W. Wang, and Z. Zhang, "Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: Association for Computing Machinery, Feb 2018, p. 269–278. [Online]. Available: https://doi.org/10.1145/3174243.3174255

[18] Q. Gautier, A. Althoff, P. Meng, and R. Kastner, "Spector: An OpenCL FPGA benchmark suite," in *2016 International Conference on Field-Programmable Technology (FPT)*, Dec 2016, p. 141–148.

[19] Achronix Semiconductor. (2019) Speedster7t FPGAs. [Online]. Available: https://www.achronix.com/product/speedster7t/

[20] M. Eldafrawy *et al.*, "FPGA Logic Block Architectures for Efficient Deep Learning Inference," *ACM Transactions on Reconfigurable Technology Systems (TRETS)*, vol. 13, no. 3, 2020.

[21] A. Boutros *et al.*, "Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2018.

[22] S. Rasoulinezhad *et al.*, "PIR-DSP: An FPGA DSP Block Architecture for Multi-precision Deep Neural Networks," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.

[23] A. Arora *et al.*, "Tensor Slices to the Rescue: Supercharging ML Acceleration on FPGAs," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021.

[24] X. Wang *et al.*, "Compute-Capable Block RAMs for Efficient Deep Learning Acceleration on FPGAs," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021.

[25] A. Arora *et al.*, "CoMeFa: Compute-in-Memory Blocks for FPGAs," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022.

[26] U. Aydonat *et al.*, "An OpenCL Deep Learning Accelerator on Arria 10," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[27] A. Boutros *et al.*, "You Cannot Improve What You Do Not Measure: FPGA vs. ASIC Efficiency Gaps for Convolutional Neural Network Inference," *ACM Transactions on Reconfigurable Technology Systems (TRETS)*, vol. 11, no. 3, 2018.

[28] S. Wang *et al.*, "C-LSTM: Enabling Efficient LSTM Using Structured Compression Techniques on FPGAs," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018.

[29] P. Whatmough *et al.*, "FixyNN: Energy-Efficient Real-Time Mobile Computer Vision Hardware Acceleration via Transfer Learning," in *Proceedings of Machine Learning and Systems*, 2019.

[30] J. Redmon. (2018) Tiny darknet. [Online]. Available: https://pjreddie.com/darknet/tiny-darknet/

[31] J. Fowers *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *International Symposium on Computer Architecture (ISCA)*, 2018.

[32] J. Ngadiuba *et al.*, "Compressing Deep Neural Networks on FPGAs to Binary and Ternary Precision with hls4ml," *Machine Learning: Science and Technology*, vol. 2, no. 1, 2020.

[33] Y. Lecun *et al.*, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[34] H. Sharma *et al.*, "From High-level Deep Neural Models to FPGAs," in *International Symposium on Microarchitecture (MICRO)*, 2016.

[35] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *International Symposium on Computer Architecture (ISCA)*, 2017.

[36] A. Vaswani *et al.*, "Attention is All You Need," in *International Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[37] S. Spanò *et al.*, "An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm," *IEEE Access*, vol. 7, 2019.

[38] L. Da Silva *et al.*, "Parallel Implementation of Reinforcement Learning Q-Learning Technique for FPGA," *IEEE Access*, vol. 7, 2019.

[39] J. Fowers *et al.*, "A High Memory Bandwidth FPGA Accelerator for Sparse Matrix-Vector Multiplication," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2014.

[40] Xilinx. (2017) Gemx. [Online]. Available: https://github.com/Xilinx/gemx

[41] Z. Wei *et al.*, "Design Space Exploration for Softmax Implementations," in *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2020.

[42] B. Darvish Rouhani *et al.*, "Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point," *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[43] S. Wang and P. Kanwar. BFloat16: The Secret to High Performance on Cloud TPUs. https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus.

[44] R. Panda and L. K. John, "Proxy Benchmarks for Emerging Big-Data Workloads," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017.

[45] S. Song *et al.*, "Proxy-Guided Load Balancing of Graph Processing Workloads on Heterogeneous Clusters," in *International Conference on Parallel Processing (ICPP)*, 2016.

[46] P. Jamieson *et al.*, "Odin II: An Open-Source Verilog HDL Synthesis Tool for CAD Research," in *2International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2010, pp. 149–156.

[47] S. A. Damghani *et al.*, "Yosys+Odin-II: The Odin-II Partial Mapper with Yosys Coarse-grained Netlists in VTR," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2022.

[48] C. Wolf, "Yosys Open SYnthesis Suite," https://yosyshq.net/yosys/.

[49] S. A. Damghani *et al.*, "Desired Footprint by Technology Mapping Modification using a Genetic Algorithm in Odin II," in *International Workshop on Rapid System Prototyping (RSP)*, 2020.

[50] G. Krylov *et al.*, "Hard and Soft Logic Trade-offs for Multipliers in VTR," in *Euromicro Conference on Digital System Design (DSD)*, 2020.

[51] E. Hung, "Mind the (Synthesis) Gap: Examining Where Academic FPGA Tools Lag Behind Industry," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2015.

[52] S. Yazdanshenas and V. Betz, "COFFE2: Automatic Modelling and Optimization of Complex and Heterogeneous FPGA Architectures," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, 2019.

[53] Arizona State University. (2012) Predictive Technology Model. [Online]. Available: http://ptm.asu.edu/

[54] Intel. (2019) Intel Agilex FPGAs and SOCs. [Online]. Available: https://www.intel.com/content/www/us/en/products/programmable/fpga/agilex.html

[55] G. Dunteman, *Principal Components Analysis*. Sage Publications, 1989.

[56] R. Panda *et al.*, "Wait of a Decade: Did SPEC CPU 2017 Broaden the Performance Horizon?" in *International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

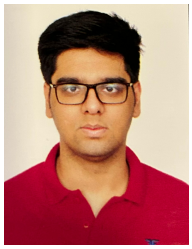
[57] M. Elgammal *et al.*, "Learn to Place: FPGA Placement Using Reinforcement Learning and Directed Moves," in *International Conference on Field Programmable Technology (FPT)*, 2020.

[58] ——, "RLPlace: Using Reinforcement Learning and Smart Perturbations to Optimize FPGA Placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021.

Aman Arora is a Ph.D. candidate in the Department of Electrical and Computer Engineering at The University of Texas at Austin. His research interests are in the areas of reconfigurable computing, domain-specific acceleration and machine learning. His work received a Best Paper Award at the IEEE FCCM conference in 2022, and he currently holds a fellowship from the UT Austin Graduate School. He has over 10 years of experience in the semiconductor industry in design, verification, testing and architecture roles. Most recently, he worked in the GPU architecture group at NVIDIA. He obtained his B.Tech. degree from National Institute of Technology, Kurukshetra, India.

Andrew Boutros received the B.Sc. degree in electronics engineering from the German University in Cairo, in 2016, and the M.A.Sc. degree in electrical and computer engineering from the University of Toronto, in 2018, where he is currently pursuing the Ph.D. degree. He was a Research Scientist at the Intel's Accelerator Architecture Laboratory, before he returned to the University of Toronto. His research interests include FPGA architecture and CAD, deep learning acceleration, and domain-specific architectures. He is an affiliate of the Intel/VMware Crossroads 3D-FPGA Academic Research Center, Vector Institute for Artificial Intelligence, and Center for Spatial Computational Learning. He received three best paper awards at Reconfig 2016, FPL 2018, and ICM 2021.

Seyed Alireza Damghani is a software engineer at Intel with a proven track record of research accomplishments, having earned an M.Sc. and B.Sc. in Computer Science. With considerable experience in designing and developing software, he has made notable contributions to the development of large-scale CAD systems. He has demonstrated his commitment to the open-source community through his work as a developer for the Verilog-to-Routing EDA tool, where he introduced the Yosys+Odin-II front end into the flow.



Karan Mathur received a bachelor's degree in Electrical and Electronics Engineering from Birla Institute of Technology and Science (BITS) Pilani, India. He worked on the Koios benchmarks as an undergraduate research intern at the University of Texas at Austin. During his time at BITS he was the student coordinator of Team Anant, a student team building a 3U CubeSat satellite with a hyperspectral camera as its payload. He is currently a professional working on design verification in the semiconductor industry in India.

Vedant Mohanty is currently a professional working in the semiconductor industry in India. He was a part-time undergraduate intern at University of Texas at Austin. He received a bachelor's degree with distinction in Electronics and Communication Engineering from Birla Institute of Technology and Science Pilani, Goa, India. His research interests include hardware acceleration for machine learning, neuromorphic computing and FPGA design.

Tanmay Anand is currently a graduate student at University of Wisconsin-Madison. Previously, he was a research intern at University of Texas at Austin from Fall 2021 to Summer 2022, during which he worked on Koios 2.0. He completed his undergraduate degree in Electrical and Electronics Engineering from Birla Institute of Technology and Science, Pilani, India. His research interests include heterogenous systems and reconfigurable computing.

Mohamed A. Elgammal is a Ph.D. candidate at the University of Toronto. His research interests include CAD tools, FPGAs, and reinforcement learning. Elgammal received the B.Sc. and M.A.Sc degrees (Hons.) in electronics engineering from Cairo University, Egypt. He is an affiliate of the Intel/VMware Crossroads 3D-FPGA Academic Research Center and Center for Spatial Computational Learning.

Kenneth B. Kent has been a Professor in the Faculty of Computer Science since 2002. He is the Barrett Chair in Entrepreneurship for Digital Transformation and the Director of the Center for Advanced Studies – Atlantic. Dr. Kent is an Honorary Professor at Hochschule Bonn-Rhein-Sieg, where he is also involved in research through the Institute for Visual Computing and the Department of Computer Science. Dr. Kent's research interests include FPGA CAD tools, parallel and distributed systems, service and cloud computing, virtual machines/runtimes and software engineering. He has published more than 200 scientific works via journals, conferences and patents.

Vaughn Betz received his B.Sc. degree in electrical engineering from the University of Manitoba in 1991, his M.S. degree in electrical and computer engineering from the University of Illinois at Urbana–Champaign in 1993, and his Ph.D. degree in electrical and computer engineering from the University of Toronto in 1998. He is the original developer of the widely used VPR FPGA placement, routing and architecture evaluation CAD flow, and a lead developer in the VTR project that is built upon VPR. He co-founded Right Track CAD to develop new FPGA CAD tools and architectures, and joined Altera upon Right Track CAD's acquisition. Dr. Betz spent 11 years at Altera (now part of Intel), ultimately as Senior Director of software engineering. He is currently a professor at the University of Toronto. He holds 101 US patents and has published over 100 technical articles in the FPGA area, fifteen of which have won best or most significant paper awards. He is a Fellow of the IEEE, the National Academy of Inventors and the Engineering Institute of Canada, and a Faculty Affiliate of the Vector Institute for Artificial Intelligence.

Lizy K. John holds the Truchard Foundation Chair in Engineering in the Department of Electrical & Computer Engineering at The University of Texas at Austin. Her research is in the areas of computer architecture, hardware architectures for machine learning, performance evaluation and benchmarking, workload characterization, and reconfigurable computing. She holds 16 U. S. patents and has published four books, 16 book chapters, 300+ refereed journal and conference publications. She is the Editor-in-Chief of IEEE Micro, and has served in the editorial boards of IEEE Transactions on Computers, IEEE Transactions on VLSI, IEEE Transactions on Sustainable Computing, IEEE Computer Architecture Letters, ACM Transactions on Architectures and Code Optimization. She is an IEEE Fellow, ACM Fellow, and Fellow of the National Academy of Inventors.