HLSDataset: Open-Source Dataset for ML-Assisted FPGA Design using High Level Synthesis

***Abstract*— Machine Learning (ML) has been widely adopted in design exploration using high level synthesis (HLS) for faster resource, timing and power estimation at very early stages for FPGA-based design. To perform prediction accurately, high-quality and large-volume datasets are required for training ML models. However, the current datasets used in this domain are proprietary or limited in use, and practitioners have to generate their own dataset to train HLS-related ML models. This paper presents a dataset for ML-assisted FPGA design using HLS, called HLSDataset. The dataset is generated from widely used HLS C benchmarks including Polybench, Machsuite, CHStone and Rossetta. The Verilog samples are generated with a variety of directives including loop unroll, loop pipeline, and array partition to make sure optimized and realistic designs are covered. The total number of generated Verilog samples is nearly 9,000 per FPGA type. The dataset repository includes CSV (comma separated values) files containing both HLS and implementation metrics which can be easily consumed by ML model. We also include original C source code with directives, Verilog designs, post-HLS reports, post-implementation reports for each sample in the dataset, so that any metrics not present in the CSV can be easily extracted. In order to extend the dataset for future benchmarks, generation and extraction scripts are also provided. To demonstrate the effectiveness of our dataset, we undertake case studies to perform power estimation and resource usage estimation with ML models trained with our dataset. All the code and dataset are public at *link-hidden-for-double-blinding*. We believe that HLSDataset can save valuable time for researchers by avoiding the tedious process of running tools, scripting and parsing files to generate the dataset, and enable them to spend more time where it counts, that is, in training ML models.**

I. Introduction

High-level synthesis (HLS) is able to convert software applications into FPGA hardware designs with different optimization strategies. It can greatly improve the productivity since hardware designers do not need to write low-level hardware description language (HDL) from scratch given an application written in a high-level language (HLL) like C, C++ or SystemC.

While HLS greatly helps to reduce the effort for the software to FPGA implementation conversion, it is quite time-consuming, especially when large design spaces need to be explored using various pragma settings. This is a common usecase when designing application-specific optimized designs targeting FPGAs, for example, when designing FPGA based accelerators for ML applications. Metrics such as resource usage and achieved clock frequency reported after HLS are estimates. To find the final metrics, the even slower implementation process (synthesis, place and route) is required. Even more efforts are needed to estimate power consumption accurately, since low-level simulation is required. For these reasons, efficient design space exploration targeting optimization of such metrics is hard. To address this challenge, machine learning (ML) based techniques are widely adopted to provide accurate resource usage and power estimation at early stage in HLS. S. Dai et al. [1] uses Lasso linear model, XGB and artificial neural network (ANN) to calibrate the resource usage and timing results from HLS reports. Graph neural networks (GNNs) and HLS reports are used to predict performance in the work by N. Wu et al. [2]. HL-Pow [3] and PowerGear [4] give solutions to predict power consumption using convolutional neural networks (CNNs) and GNNs respectively. E. Ustun et al. [5] builds graph samples using the IR (intermediate representation) generated during HLS and use them as input to GNNs to predict operation delay.

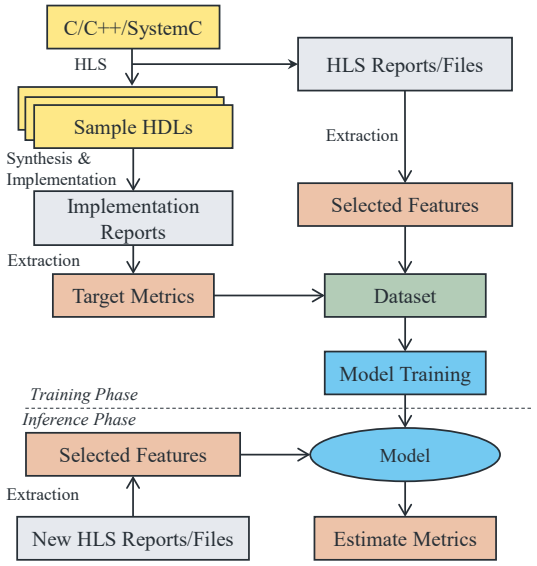

Fig. 1: The flow of general ML-based methods in HLS

The flow of general ML-based methods in HLS domain is shown in Fig 1. ML based methods can provide fast and accurate metrics estimation with HLS reports, however, extensive dataset is needed to train the models to produce acceptable results. To generate task-specific dataset in HLS domain requires lots of effort:

- Software source code should cover enough domains
- Source code should be well manipulated with HLS directives so that HLS optimization can be applied
- Varieties of optimization strategies need to be applied to the source code so that wide range of hardware designs can be

generated
- Implementation is needed if the post-implementation metrics are the prediction goal
- Extensive scripting is required to extract the data from reports and preprocess before it can be consumed directly in ML models
- Significant computing resources may be needed for large number of tool runs to collect enough data

Researchers have to generate their own dataset, which can be extremely time-consuming because of the aforementioned reasons. Due to the different prediction goal and ML models, existing datasets are proprietary and not shareable or reusable. However, there is an opportunity to reduce, and even eliminate, the redundant work for various researchers by creating a dataset that contains common usable information, allowing them to focus on training the ML models instead of generating the dataset. We observe that resource usage reports, Intermediate Representation (IR) code, IR operator information, Finite State Machine Data path (FSMD) model from HLS are commonly used as the source of features. The resource utilization, timing information and power consumption values from post-implementation phase are the common metrics that researchers are interested to predict.

With the above observation, we propose HLSDataset: a well-curated open-source dataset for ML-assisted FPGA design using HLS. Our dataset can be used by a large subset of problems in this domain. The dataset currently contains nearly 9,000 Verilog designs per FPGA type, and two FPGA types are covered. To ensure diversity of designs, HLSDataset are generated from multiple applications across various benchmarks: Polybench [6], Machsuite [7], CHStone [8] and Rosetta [9], and each application is tuned to generate a variety of hardware design samples. Our dataset contains all necessary files and reports for every design (or, sample) so that features and target metrics can be easily extracted. In this paper, we describe the dataset, how it can be used, and showcase its utility by conducting two case studies. We expect this dataset to be widely usable and get even more useful with time through contributions by the FPGA research community.

Our contributions in this paper are as follows:

- Introduce HLSDataset and describe both the properties and usage of the dataset.
- Present the methodology how HLSDataset is generated. This methodology can be easily replicated to extend the dataset.
- Two case studies are conducted to demonstrate the effectiveness of HLSDataset.

Our dataset (including C code, Verilog code, CSV files, reports, and scripts) is open-sourced at *link-hidden-for-double-blinding*. The rest of this paper is organized as follows: Section II summarizes the existing datasets and compares our dataset with them; Section III illustrates the methods we use to generate HLSDataset; Section IV describes the contents of HLSDataset; Section V gives a general overview of where HLSDataset can be used; Section VI presents two case studies that use the dataset to successfully accomplish the prediction tasks, followed by a summary of this work and future work in Section VII.

II. Related work

The success of ML-based models depends on well-curated datasets. There are a few datasets for training ML models to assist in chip design problems in the ASIC domain. OpenABC-D [10] from NYU is a large-scale, labeled dataset produced by synthesizing open source designs with an open-source ASIC logic synthesis tool. This dataset can be used in developing, evaluating and benchmarking ML-guided logic synthesis but is applicable to a very small subset of problems i.e. prediction of ASIC synthesis results. CircuitNet [11] is another open-source dataset targeted for three prediction tasks in backend ASIC flows - congestion prediction, DRC (Design Rule Check) violation prediction, and IR drop prediction. It contains more than 10000 samples (in form of 2D image-like data) obtained by running open-source RISC-V designs through commercial EDA tools. This dataset is applicable to only a few physical design problems.

For FPGA HLS design flow, which is the focus of this paper, there are a few open-source datasets as well. Dai et al. [1] have open-sourced a dataset that is applicable to prediction of resource usage and delay (or frequency) for FPGAs from high-level applications written in C. The dataset is generated by using applications from suites such as CHStone, Machsuite and Rosetta, and the Vivado tool chain from Xilinx/AMD. This dataset is restricted to use only in estimation of resource usage and timing for FPGA, and contains only limited data. The data provided is only for 1 FPGA device, implying that this dataset can not be used for cross-FPGA predictions.

MLSBench [12] is an open-source dataset generated from 17 C/C++ and 13 SystemC benchmarks using Xilinx Vivado HLS tool flow. The C sources to generate the designs are from S2CBench [14], CHStone [8] and MachSuite [7]. The dataset contains only log files and reports generated from Xilinx Vivado HLS tool flow, but without directly consumable features, labels and RTL codes. Also, this dataset is limited to only one FPGA. Therefore, MLSBench is hard to extend and quite limited in ML usage.

Spector [13] is a benchmark suite that contains applications written in OpenCL. The authors run the benchmarks through Intel OpenCL SDK to generate 8300 hardware designs targeted for Intel FPGAs. In addition to just the benchmarks, several metrics for each design sample (based on compilation using Intel OpenCL SDK) are also provided. The focus is on HLS tool flows and design space exploration.

Table I compares the various properties of these datasets. We show the number of samples contained in the dataset and number of sources used for generating the dataset. These datasets generally cater to limited usecases (eg: physical design prediction in [11], or RTL synthesis quality prediction in OpenABC-D [10] or resource usage prediction in Dai et al.[1]). Some need further expansion and curation to be readily usable by others. Retargeting the few available datasets for a new ML model requires significant manipulation and

Work	# Samples	# Sources	Platform & Abstraction level	Tools	Use Case in ML
OpenABC-D [10]	870,000	29	ASIC RTL	OpenROAD	Estimation of quality of a synthesis recipe
CircuitNet [11]	12,960	6	ASIC Physical Design	Synopsys DC	Congestion prediction, DRC violation Prediction, IR drop prediction
Dai [1]	1,300	65	FPGA HLS	Xilinx Vivado	Quality of Results Estimation on one FPGA
MLSBench [12]	6,000	30	FPGA HLS	Xilinx Vivado	NA
Spector [13]	8,300	9	FPGA HLS	Altera OpenCL SDK	NA
Ours	18,876	34	FPGA HLS	Xilinx Vivado	Power Estimates, resource and timing estimation, operation delay estimate, cross-FPGAs studies, and more

TABLE I: Comparing HLSDataset with prior open-source datasets for training ML models for chip design

augmentation. So, researchers often generate their own dataset every time they want to solve a new problem. In this process, they have to rerun tool flows to generate reports and then write scripts to parse those reports repeatedly. This motivates us to develop a dataset that is retargetable, versatile and robust, so that researchers do not need to replicate the tedious process of generating the dataset.

We focus on developing a dataset for predictions from applications written in high-level languages (HLLs) because high-language models of applications are available in early stages of development of customized designs such as application-specific accelerators. In other words, we focus on prediction at the HLS level. Predicting at the HLS level provides the most benefit in design space exploration. We present HLSDataset, an open-source dataset for ML-Assisted FPGA Design for HLS.

III. HLSDataset Construction

Table II gives a general overview of our HLSDataset. We use HLL sources belonging to various application domains such as multimedia, arithmetic, signal processing and machine learning, from multiple popular benchmark suites such as Polybench [6], Machsuite [7], CHStone [8] and Rosetta [9]. Xilinx Vivado/Vitis tool chains are used for HLS and implementation. Two FPGAs are used: ZU9EG and XC7V585T. We plan to expand the dataset to include more FPGAs, including Intel FPGAs. One target frequency of 100 MHz is used. We are working on using more target frequencies as well.

Category	Details
Num samples	18,876
Num applications	34
Application sources	Polybench, Machsuite, CHStone, Rosetta
FPGAs	ZU9EG, XC7V585T
Clock frequency	100MHz
Domains	Multimedia, Arithmetic, Signal processing, ML
Size	50 GB
Machines	9 16-core Intel Xeon 5218 2.3GHz 384 GB RAM
Time	More than 1,500 hours
Tools	Xilinx Vivado/Vitis

TABLE II: General overview of HLSDataset

A. C source code manipulation

Verilog designs generated from C benchmarks are highly dependent on HLS directives, pragmas and the target clock frequency. For generating our dataset, we focus on the design space of *loop unroll*, *loop pipeline* and *array partition*. Loops in C code need to be labelled so that loop unroll and loop pipeline can be applied to generate efficient designs. Machsuite and Rosetta are already well-written with HLS directives, and we directly use their code for our dataset generation. We manipulate the Polybench and CHStone source code with HLS directives.

B. Auto-generation of Tcl scripts

The scope of generated Verilog designs can be huge, since the factors for *array partition* and *loop unroll* can vary greatly. The number of generated designs is determined by the number or the dimension of the factors we want to explore in our dataset. However, manually writing every Tcl script (Xilinx Vivado/Vitis tools use a Tcl script based interface), which is used to tune HLS solution for the generation of Verilog designs in our dataset, is time-consuming and unrealistic. In order to generate designs more efficiently, we write a template Tcl script for every C source code and a script to parse it. The script will auto-generate Tcl files which can be directly used by the HLS tool.

An example *template.tcl* is shown in Fig 2. It contains 4 blocks of lines which are classified into three types: static lines, array partition lines and loop optimization lines.

- Static lines: The directive lines under static lines are not subject to change, they should be the same and written into every generated Tcl file.
- Array partition lines: The first line indicates the sets of parameters applied for HLS. It contains a number denoting the number of directive lines, a list of numbers denoting the factor sets for array partition and a set of types for array partition. The rest of lines are the directive lines with placeholder that should be replaced with the parameters defined by the first line. The placeholders inside the directive lines are replaced with the combination of factor sets and type sets, and every Tcl file will contain one combination of directive parameter. The array partition lines 2 in the Fig 2 generates 8 combination directive parameters in this case

due to 4 factors and 2 types. Note that factor equalling to 1 means no array partition is applied.

- Loop optimization lines: The first line denotes the number of nested loops and the number of directive lines for loop optimization. It is followed by loop optimization parameter lines, each of which indicates the depth of a loop, the name of a loop, whether to apply pipeline to the loop, whether to apply unroll to the loop and unroll factor sets. The rest of the lines are directive lines with placeholders that should be filled with settings from the loop optimization parameter lines. Unroll and pipeline are applied to at most one layer of nested loops, therefore, the number of generated directives is equal to the sum of the number of unroll factors among all the loops and the one without any loop optimization. The loop optimization lines in Fig 2 generates 8 combination directives for the loop optimization.

```
#static lines
set_directive_resource -core RAM_1P_BRAM "bfs" nodes
set_directive_resource -core RAM_1P_BRAM "bfs" edges

#array partition lines 1, factor dimension = 6
array_partition,2,[1 2 4 8 16 32],[cyclic]
set_directive_array_partition -factor [factor] -type [type] "bfs" nodes
set_directive_array_partition -factor [factor] -type [type] "bfs" edges

#array partition lines 2, factor dimension = 4*2 = 8
array_partition,2,[4 8 16 32],[cyclic block]
set_directive_array_partition -factor [factor] -type [type] "bfs" levels
set_directive_array_partition -factor [factor] -type [type] "bfs" level_counts

#loop optimization lines 1, factor dimension = 8
loop_opt,2,2
0,loop_horizons,,unroll,[2 5 10]
1,loop_nodes,pipeline,unroll,[2 4 8 16]
set_directive_pipeline bfs/[name]
set_directive_unroll -factor [factor] bfs/[name]
```

Fig. 2: Example template Tcl file to generate the optimization strategy for the application `bfs` from Machsuite

The blocks of directive lines are independent of each other, therefore the number of Tcl files is equal to the products of the number of directive parameter combination among all the blocks. In this example template, 384 Tcl files are generated, and different optimization strategies are expected. The method to generate multiple versions of Tcl files is summarized in **Algorithm 1**, each block of lines will be parsed into an object.

C. Data collection

IR code, IR operator information, FSMD model files from HLS and resource utilization reports from both HLS and implementation are included in our dataset. In order to get the high-confidence power estimation, we write testbench and run post-implementation functional simulation for vector-based power estimation.

We observe that there is a chance that the HLS tool generates the same design even though different optimization strategies are provided in the Tcl script. This can be caused by aggressive optimization parameters, which are identified as unachievable by the HLS tool. The tool then automatically downgrades the optimization parameters, which can match optimization parameters during another run. Therefore, redundant designs can be generated. We identify redundant designs by checking the hierarchical resource utilization from HLS reports. If two or more designs have exactly the same utilization, only one will be kept in our dataset.

Algorithm 1: Method to generate multiple Tcl files

Input: template.tcl
Output: N different versions of Tcl files
$s\_lines, array\_objs, loop\_objs$ from $template.tcl$;
Generate N empty Tcl files
`/* static lines for each Tcl file */`
for $i \leftarrow 1$ **to** N **do**
  Write s_lines to Tcl file
end
`/* array partition directives */`
foreach $o \in array\_objs, f \in o.factors, t \in o.types$ **do**
  $array\_partition$ with factor f and type t
  Write $array\_partition$ to Tcl file
end
`/* loop unroll and pipeline */`
foreach $o \in loop\_objs, f \in o.factors$ **do**
  Get the $loop$ from $loop\_list$ in o
  Apply $pipeline$ to $loop$ if pipeline applies
  Apply $unroll$ to $loop$ with factor f if unroll applies
  Write $pipeline$ and $unroll$ to Tcl file
end

IV. Properties of HLSDataset

A. The contents of HLSDataset

HLSDataset contains nearly 9,000 hardware design samples for each FPGA type and we consider the features listed below to sufficiently characterize each design sample:

1) Resource usage (the number of BRAM, DSP, FF and LUT)
2) Application domain (e.g., video/graph processing, linear algebra etc)
3) The number of arithmetic operators (e.g., add, mul), the number of logic operators (e.g., or, shift)
4) The number/size of primary inputs and outputs
5) The number of registers, memory and multiplexers
6) Clock period

Power consumption is also included, since it is crucial when low-power hardware designs are the final target. We preprocess the raw reports and files from both HLS and implementation phases and generate two CSV files for each benchmark. Each CSV file contains multiple entries depending on the number of generated hardware designs for the benchmark. The user can directly use the data in the CSV files to train their ML models, thereby avoiding any effort in changing source code, setting up and running tools, and parsing reports. The detailed contents of the CSV files are listed in Table III.

It is possible that the features that other researchers are interested in, are not present in the CSV files. Therefore, we also create tar balls containing all the necessary files for feature extraction to do ML training. These files are selected according to how prior works generate their own dataset. Each tar ball contains:

- Generated Verilog code (*.v)

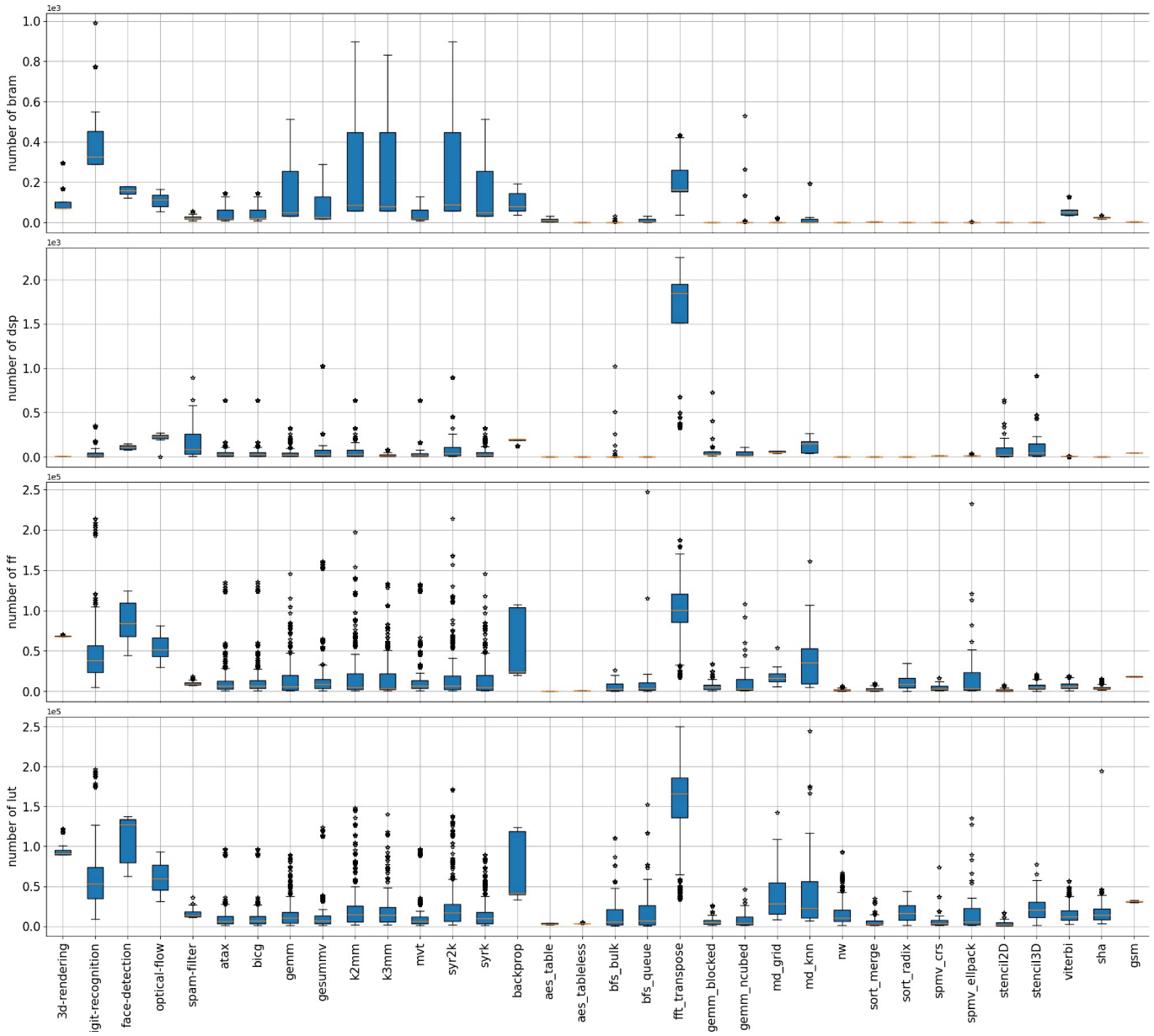

Fig. 3: Resource utilization of designs generated for ZU9EG, applications are from Rosetta, Polybench, Machsuite and CHStone

- IR code (**.bc*)
- IR operator information (**.adb*)
- FSMD model (**.adb.xml*)
- Resource usage estimation from HLS (**.verbose.rpt* and **.verbose.rpt.xml*)
- Resource utilization reports (*utilization.xls*) and timing reports (*timing.xls*) generated after implementation

Considering the reusability and ease-of-use of the dataset, Tcl scripts and source code files are included in the dataset so that researchers can easily extend the dataset with other benchmarks. The detail of how the Tcl script templates can be used is discussed in Section III. We also include Verilog testbenches so that the generated designs can be easily evaluated with simulation-based power estimation.

Overall, the contents of HLSDataset are summarized as:

1) The CSV files containing features for each hardware design listed in Table III
2) Tcl templates and actual scripts to generate the dataset
3) C source code files manipulated with HLS pragmas
4) Testbenches in Verilog to test generated Verilog designs
5) Tar balls containing raw files and reports from HLS and reports from implementation stage

Compared to the datasets used by prior works, HLSDataset gives a wider coverage of information for each design, and it gives higher chance for researchers to use or extract useful features directly, meanwhile, no efforts are needed to run the

Category	*post_hls_info.csv* description
Resource #	Estimated usage and available number of BRAM, DSP, FF and LUT
Clock	Target, estimation and uncertainty of the clock period
Logic ops	The number of C and RTL logic operators and associated resource usage
Arith ops	The number of C and RTL arithmetic operators and associated resource usage
Data ports	Width and the number of data input and output ports
Category	***post_implementation_info.csv* description**
Power	Simulation-based dynamic power consumption
Resource #	Actual usage of BRAM, DSP, FF and LUT
Clock	Achieved clock frequency

TABLE III: Descriptions of features included in the CSV files provided with HLSDataset

time-consuming HLS and implementation tool flows.

B. Statistical overview of HLSDataset

Fig 3 provides a view of the diversity of the HLSDataset, through the resource usage metrics of the designs (or samples) contained in the dataset. We use box and whisker plots to show the distribution of LUTs (Look Up Tables), FFs (Flip Flops), DSPs (Digital Signal Processing Blocks) and BRAMs (Block RAMs) consumed by the designs generated from each application. As mentioned earlier, we use 4 widely used benchmark sets - Polybench, Machsuite, CHStone and Rosetta - to generate our dataset. Machsuite and Polybench are mainly composed of short programs and kernels, however, tuning the directives aggressively can still lead to large resource usage on FPGA. Rosetta, on the other hand, is composed of applications from ML and image or video processing domains. Each application of Rosetta contains multiple kernels, and it leads to larger resource usage on FPGA. The secure hash algorithm SHA and linear predictive coding analysis GSM are picked from CHStone due to their representative in the domain. We chose not to include arithmetic operation programs from CHStone due to the limited HLS design space in those applications.

V. HLSDataset Applications

HLSDataset can be applied to a multitude of prediction applications. Table IV summarizes the prior works in the area of prediction at the HLS level. The data required for training ML models for each of these prior works is included in HLSDataset. Hence, HLSDataset can be effectively used for these and similar works.

Resource utilization estimates: HLSDataset can be directly used for post-implementation resource utilization estimates. Dai et al. [1] use Lasso linear model, XGB and artificial neural network (ANN) to improve the quality of HLS-generated resource utilization values with features extracted from HLS reports. Wu et al. [2] predict post-implementation resource usage by using the graph structure obtained from the IR codes generated by front-end of HLS tools. Fast estimation of resource usage find application in design space exploration while generating overlay architectures for FPGAs [15]. The features and feature source used to conduct the studies can be easily found and extracted from HLSDataset.

Timing and operation delay prediction: Wu et al. [2] demonstrates prediction of post-implementation critical path timing using IR codes and features from HLS reports. D-SAGE [5] builds graph samples using the IR generated during HLS and use them as input to GNNs to predict operation delay. HLSDataset contains the IR code files as well as HLS reports generated by Vivado HLS, and can be used to train such models to predict timing related information.

Power estimates: HL-Pow [3] and PowerGear [4] train ML models to predict power consumption using convolutional neural networks (CNNs) and GNNs respectively. Predicting power consumption needs data such as signal activities and operators obtained from the IR. While those signal activities are not directly included in HLSDataset, testbenches and stimulus are provided so that both RTL-level simulation and C-level simulation can be conducted. Necessary codes to run the simulation: IR codes and RTL designs are included in HLSDataset.

Beyond the above tasks, HLSDataset can be applied to many more potential use cases. While the mentioned works target single-FPGA prediction, HLSDataset includes samples from multiple FPGAs. We believe HLSDataset has the potential to be used for cross-FPGAs metric prediction, although no existing work shows this usecase. In addition to prediction of results, HLSDataset can be used to train models to optimize EDA tools and help on faster design space exploration. Moreover, HLSDataset can also be used to evaluate the ML model efficiency in HLS domain with the advancing of ML techniques.

The features and labels used by each ML models vary widely depending on the task and ML algorithm used, as we can see from table IV. By including information from different levels in the CSV files and TAR balls in HLSDataset, we ensure that all such ML models can be trained. Researchers can extract information from TAR balls and apply HLSDataset to many other applications.

VI. Case studies

Our dataset covers large amount of features and metrics from post-HLS and post-implementation reports which can be used in machine learning models directly. Therefore, users can simply extract the necessary information from our dataset to train and test their models. In this section, we perform two case studies by training and testing ML models with HLSDataset to demonstrate the usage of it.

A. Case Study 1: Power Estimation in FPGA HLS via GNNs

In our first case study, we replicate the graph neural networks (GNNs) in PowerGear [4] to predict the post-implementation power using both post-HLS features and signal information extracted from C-level simulation. We used simulation power as our ground truth power. The GNN models are trained and tested with the subset of HLSDataset on the

Work	ML model	Task	C source	Feature and source
[1]	Lasso, XGB, ANN	Resource usage and timing	CHStone, Machsuite, S2CBench, Rosetta	Resource usage estimation for logic ops, arithmetic ops, memory and multiplexer; achieved clock period and uncertainty from **HLS reports**
[2]	GNN	Resource usage and timing	CHStone, Machsuite, Polybench	Graph samples based on **IR code**; operator type, used resource type and timing information from **HLS reports**
[3]	CNN	Power estimates	Polybench	Resource utilization and clock estimation by **HLS reports**; signal activities track and IR operator information from **IR code**; RTL operator information from **FSMD model**
[4]	GNN	Power estimates	Polybench	Signal activities track and IR operator information from **IR code**, Graph samples built with **IR code** and **FSMD model**, RTL operators information in **FSMD model**
[5]	GNN	Operation delay	Machsuite	Graph structures, operation type and bitwidth from **IR code**

TABLE IV: Prior ML-based prediction via HLS work, the used ML model, prediction tasks, the used dataset for training and the availability of the dataset.

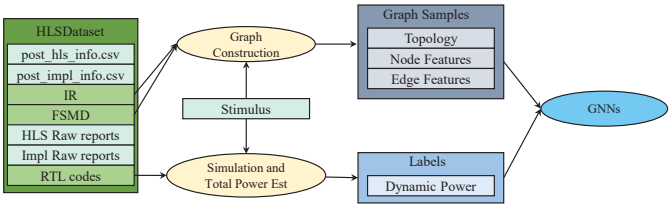

Fig. 4: Usage of HLSDataset to construct machine learning based power model

same FPGA. IR code can be directly used to build graph samples which serve as the inputs to the GNNs. The usage of HLSDataset in this case study is shown in Fig 4.

The model is trained using the dataset from Polybench. We leave one target application out of the nine application as the test dataset and use all the rest for training. With the iteration of the approach, we generate one model for every application. We perform 10-fold cross-validation for model generation. All the above steps are repeated for the dataset from the other FPGA. All the training and testing run on Nvidia Ampere A100 GPU. The results for two FPGA devices, ZU9EG and XC7V585T, can be found in Table V. The test errors for dynamic power range from 3.89% to 7.93% on ZU9EG and from 5.25% to 9.43% on XC7V585T, and the average errors are 5.08% and 6.40% respectively. The accuracy results are better than what PowerGear presents since we use simulation-based power as ground-truth value while they use on-board measurement power. The results show that HLSDataset can be used to perform ML-based power estimation tasks for FPGA.

B. Case Study 2: Estimation of Quality of Results in HLS with ML

The resource usage estimation (LUTs, FFs, DSPs, BRAMs) generated by HLS tools are fast but inaccurate compared to the post-implementation reports because HLS tools simply sum up the contributions of instantiated functional units during the synthesis. This approach fails to capture the optimization effects and limitations imposed by resources on-chip. However, as S. Dai et al. [1] indicates, ML can help to predict more-accurate resource usage from estimates in the HLS reports.

Application	Error of Dynamic Power (%)	
	ZU9EG	XC7V585T
atax	3.89	5.25
bicg	3.90	5.60
gemm	5.24	6.50
gesummv	7.93	9.43
k2mm	4.25	6.00
k3mm	4.15	6.47
mvt	4.64	5.62
syrk	5.31	6.22
syr2k	6.41	6.46
average	**5.08**	**6.40**

TABLE V: **Dynamic power estimation errors** - Training dataset and testing dataset are from Polybench subset of HLSDataset. Results for ZU9EG and XC7V585T.

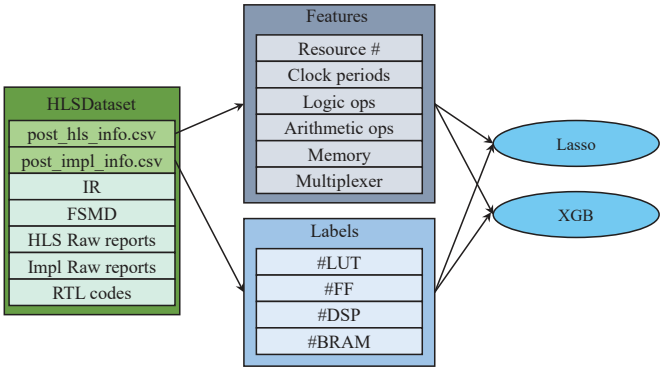

Fig. 5: Usage of HLSDataset to construct machine learning model for estimation of resource utilization

We replicate the ML model but use our HLSDataset as training and test set to evaluate the ML model on estimation of post-implementation resource usage. The way to use HLSDataset is illustrated in Fig 5. Machsuite, Polybench subsets from HLSDataset are used to train the XGB and

Lasso linear model. The features are extracted from FSMD file (.adb.xml) and resource estimates reports (.verbose.rpt.xml). The ground-truth resource utilization is extracted from post-implementation reports. All the files and reports are included in our dataset, only a parser is needed to extract necessary data to be used in the ML model. Single-task XGB and Lasso model are used in our case. We randomly select 20% of 8735 samples from the subsets as the testing set and the rest as the training/validation test set. 10-fold cross-validation is performed during training, and 75% of the training/validation set is selected for training and 25% for validation. The results are shown in Table VI. The HLS tool fails to provide good estimates for LUT and FF usage, while DSP and BRAM estimates are accurate. XGB and Lasso demonstrate a significant accuracy improvement in the estimation of LUT and FF usage. The results shown in this table differ from those in the original paper because there are differences in target FPGA, the dataset, features used to train the model and the version of HLS tools used for the dataset generation. Therefore, we do not show a comparison with the original work here.

Resource	LUT	FF	DSP	BRAM
HLS Estimate	63.2%	34.1%	0.0%	1.8%
XGB	3.2%	2.3%	NA	0.1%
Lasso	13.2%	15.4%	NA	NA

TABLE VI: **Resource estimation errors** - Training dataset and testing dataset are from Machsuite and Polybench subsets of HLSDataset. Results for ZU9EG.

VII. Conclusion

This work presents HLSDataset, a dataset for ML-assisted FPGA design using HLS. HLSDataset covers a wider range of data than other datasets in this domain, and is the first open-source dataset of its kind that can be used for multiple studies. We demonstrate that HLSDataset can be used in training ML models targeting different applications such as resource usage prediction, power prediction, etc. We also present the methodology to generate the dataset so that HLSDataset can be futher extended.

We are currently expanding HLSDataset by including data for more target frequencies (e.g. clock period = 5ns, 2.5ns, etc.). For future work, we plan to extend HLSDataset to include more benchmarks (e.g., S2CBench). and more FPGAs (including Intel FPGAs). While the design samples in HLS-Dataset are generated from C benchmark so that ML-assisted HLS based studies can be conducted, we plan to extend the dataset to include data from native Verilog benchmarks so that ML-assisted Verilog based studies are possible with our dataset.

References

[1] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Young, and Z. Zhang, "Fast and Accurate Estimation of Quality of Results in High-Level Synthesis with Machine Learning," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 129–132.

[2] N. Wu, H. Yang, Y. Xie, P. Li, and C. Hao, "High-Level Synthesis Performance Prediction Using GNNs: Benchmarking, Modeling, and Advancing," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 49–54. [Online]. Available: https://doi.org/10.1145/3489517.3530408

[3] Z. Lin, J. Zhao, S. Sinha, and W. Zhang, "HL-Pow: A Learning-Based Power Modeling Framework for High-Level Synthesis," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 574–580.

[4] Z. Lin, Z. Yuan, J. Zhao, W. Zhang, H. Wang, and Y. Tian, "PowerGear: Early-Stage Power Estimation in FPGA HLS via Heterogeneous Edge-Centric GNNs," in *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe*, ser. DATE '22. Leuven, BEL: European Design and Automation Association, 2022, p. 1341–1346.

[5] E. Ustun, C. Deng, D. Pal, Z. Li, and Z. Zhang, "Accurate Operation Delay Prediction for FPGA HLS Using Graph Neural Networks," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3400302.3415657

[6] L.-N. Pouchet, "Polybench: The polyhedral benchmark suite," 2012. [Online]. Available: http://www.cs.ucla.edu/%7Epouchet/software/polybench

[7] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, 2014, pp. 110–119.

[8] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "CHStone: A benchmark program suite for practical C-based high-level synthesis," in *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2008, p. 1192–1195.

[9] 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

[10] A. B. Chowdhury, B. Tan, R. Karri, and S. Garg, "Openabc-d: A large-scale dataset for machine learning guided integrated circuit synthesis," no. arXiv:2110.11292, Oct 2021, arXiv:2110.11292 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2110.11292

[11] Z. Chai, Y. Zhao, Y. Lin, W. Liu, R. Wang, and R. Huang, "Circuitnet: an open-source dataset for machine learning applications in electronic design automation (eda)," *Science China Information Sciences*, vol. 65, no. 12, pp. 227 401, s11 432–022–3571–8, Dec 2022.

[12] P. Goswami, M. Shahshahani, and D. Bhatia, "Mlsbench: A benchmark set for machine learning based fpga hls design flows," in *2022 IEEE 13th Latin America Symposium on Circuits and System (LASCAS)*, Mar 2022, p. 1–4.

[13] 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.

[14] B. C. Schafer and A. Mahapatra, "S2CBench: Synthesizable SystemC Benchmark Suite for High-Level Synthesis," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 53–56, 2014.

[15] S. Liu, J. Weng, D. Kupsh, A. Sohrabizadeh, Z. Wang, L. Guo, J. Liu, M. Zhulin, R. Mani, L. Zhang, J. Cong, and T. Nowatzki, "OverGen: Improving FPGA Usability through Domain-specific Overlay Generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Chicago, IL, USA: IEEE, Oct 2022, p. 35–56. [Online]. Available: https://ieeexplore.ieee.org/document/9923882/