A design hackathon to bridge AI and hardware

Hideharu AMANO, Takao GOTO
Mizuho NITAMI, Yuki MITARAI
Jiawei YU, Yuxuan PAN
Atsutake KOSUGE, Makoto IKEDA
System Design Lab, Graduate School of Engineerig, The University of Tokyo

Abstract—

This paper introduces a design hackathon approach and case study that allows beginners in hardware design to compete in terms of performance and accuracy using YOLO, a representative object detection method. Rather than optimizing the hardware itself, the focus is on employing system-level optimization techniques on a low-cost KV260 board. Even students from non-technical backgrounds were able to achieve several-fold performance improvements with the help of tools like ChatGPT.

I. Introduction

The semiconductor industry has recently come into the spotlight due to massive investments in fabs, but it still suffers from a shortage of skilled personnel. The semiconductor field requires a broad range of knowledge and experience, including system design, circuit design, physical design, and manufacturing processes, making it difficult to develop talent in a short period of time. Above all, the most important task is to increase the number of students.

The "Agile-X: Platform for the Democratization of Innovative Semiconductor Technologies" project [1] was launched in 2022 as part of the Ministry of Education, Culture, Sports, Science and Technology's initiative to establish the next-generation integrated circuit center (X-NICS). As part of its efforts to increase the semiconductor design population by a factor of ten, the project carries out a variety of educational and research activities, including the organization of design hackathons.

In this hackathon, participants implement image recognition—one of the most common applications of AI—on FPGA (Field Programmable Gate Array) boards. This allows them to understand that such processing is executed on semiconductor chips, and to explore methods for improving both performance and accuracy. Since 2023, this approach has been adopted as a graduate-level course at the University of Tokyo. It has since been refined and expanded, with supporting materials developed in the form of a wiki and blog, and is now being made available to the broader public.

Student experiments using FPGAs are widely utilized in hardware and system education because they are easy to set up and have minimal running costs. However, many of these experiments require students to understand HLS (High-Level Synthesis) or HDL (Hardware Description Language) and directly use FPGA design tools such as Vitis and Vivado. In such cases, implementing AI applications like image recognition demands a substantial amount of prerequisite knowledge.

To address this, our hackathon used the KV260 board while abstracting away the standard FPGA design processes. Participants were instead encouraged to compete in terms of processing speed and recognition accuracy through system-level innovations. As a result, even students without prior design experience, including those from non-technical (liberal arts) backgrounds, achieved more than a twofold improvement in performance.

The rest of the paper is organized as follows: Section 2 surveys conventional FPGA design contests and what's new in our trial. Section 3 describes the flow of our hackathon, and Section 4 summarizes the trial of 2024. Section 5 is for conclusions and ongoing activities.

II. Survey

A. AI Edge Contest

The AI Edge Contest, organized by Japan's Ministry of Economy, Trade and Industry (METI), focuses on image recognition algorithms. The second contest in this series served as a reference when we launched our own hackathon [2]. However, with each iteration, the challenges have become increasingly advanced. The theme of the most recent sixth contest involved developing an algorithm for 3D object detection using both forward-facing vehicle images and omnidirectional point cloud data.

A distinctive feature of this contest is the requirement to use RISC-V for part of the actual processing. In addition to the Ultra96 board, any FPGA board or RISC-V-equipped board is eligible. Participants are allowed to implement on soft-core RISC-V processors on PolarFire or Ultra96 boards, or even connect a standalone RISC-V board to an FPGA board.

The contest provides forward-facing vehicle image data, full-surround point cloud data, and labeled 3D bounding

boxes with object categories. From this data, the goal is to recognize passenger vehicles and pedestrians.

Teams of up to five members participate by submitting their source code, area report, implementation approach, and performance results. Award-winning teams are selected through a judging process. The contest is well-sponsored, with prize money of 1 million yen, 600,000 yen, and 300,000 yen awarded to the top three teams, respectively. In the 2023 contest, there were 41 entries and 271 participants.

B. FPGA Design Contest — Aiso Cup

The FPGA Design Contest (Aiso Cup) is organized by the Technical Committee on Reconfigurable Systems of the Institute of Electronics, Information and Communication Engineers (IEICE). Both an international and a domestic contest are held under the same theme. While the international contest was traditionally part of the ICFPT event, in 2024 it will be held in conjunction with ICCE in Las Vegas, and the same arrangement is planned for 2025. The "Aiso Cup" title applies only to the domestic contest.

The domestic contest serves as a preparatory stage for the international competition; therefore, it is not being held in the current year. Initially, the contest theme involved connecting FPGA boards via serial cables to compete in hardware-based game AI battles. More recently, the format has shifted to using small autonomous vehicles equipped with FPGAs, which navigate a designated course using only camera input. Tasks include obstacle avoidance, traffic signal recognition, and pedestrian protection.

In this contest, while there are no restrictions on the type of FPGA used, physical size limitations of the vehicle effectively limit the hardware to smaller models. Although the vehicle's capabilities do influence the outcome, the contest is primarily focused on FPGA design, and support such as vehicle lending is provided.

The contest takes place over two days, and participants must bring their vehicles to the event in person. Controlling the vehicle solely through camera input, without any additional sensors, presents a significant challenge. However, in recent years, some teams have successfully completed all tasks. Most participants are university lab teams, and there is no limit on the number of participants per team. Typically, around ten teams take part, though participation has become somewhat fixed in recent years. Additionally, the operational burden of running the contest largely falls on Okayama University.

C. Intel InnovateFPGA Design Contest

The Intel InnovateFPGA Design Contest is an FPGA design competition hosted by Intel, branded as a global design contest aimed at inspiring teams to develop FPGA-based projects with the theme of sustainability [3]. Projects that have been awarded include large-scale initiatives such as coral reef restoration and cloud-based management of solar converters.

A key component of the contest is the integration of cloud and edge computing using the FPGA Cloud Connectivity Kit, which is certified for Microsoft Azure and enables cloud connectivity. Although the FPGA itself (Cyclone series) is not particularly powerful, heavy processing is handled in the cloud.

This is a large-scale contest with many sponsors and a total of 260 participants. It is a highly unique competition with a strong focus on cloud-edge collaboration, but it has only been held once, in 2022.

III. Our design hackathon

In addition to the FPGA design contests introduced in Section 2, many universities also conduct FPGA design in a hackathon format as part of student laboratory exercises. However, most of these are integrated with courses on logic circuit design or computer architecture, and they focus on evaluating students' hardware design skills or system design capabilities, including processors. While effective for technical training, these approaches often require substantial prerequisite knowledge and are not well suited for helping students who are interested in AI understand that "AI is fundamentally supported by semiconductor chips."

Moreover, extending these methods to reach engaging AI applications raises the barrier even higher.

To address this, we employed Vitis-AI on the KV260 SoM to abstract away the hardware design aspect of FPGAs, allowing students to implement practical, real-world applications on FPGAs. In this hackathon, participants were challenged to improve performance and accuracy based on their creativity, without making any modifications to the underlying FPGA hardware. The methodology is described in detail below.

A. KV260 SoM Board and Vitis-AI

The KV260 is one of the Kria K26 series and is a compact board equipped with a Zynq UltraScale+ device [4]. It features rich I/O options and 4GB of DRAM, and supports secure communication with Windows PCs via TPM 2.0 (Trusted Platform Module). It includes slots for both primary boot memory and a secondary boot SSD.

The KV260 is referred to as a SoM (System-on-Module), and its design environment is built to treat the entire board as a complete system. As a result, detailed information such as the exact chip model of the Zynq UltraScale+ and its pin assignments is not publicly disclosed.

Design information for the board is integrated into Xilinx's design tools, Vitis and Vivado. In other words, the FPGA on the KV260 is not used in isolation, but is managed as part of the entire board through these tools.

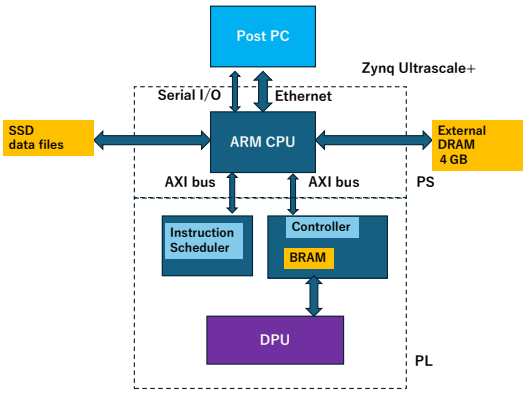

Fig. 1. Hardware Structure used in Vitis AI

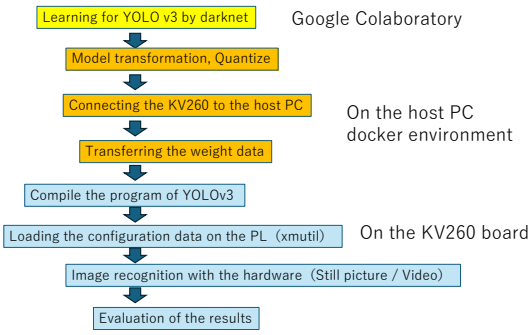

Fig. 2. Steps for Implementing YOLO v3

Vitis-AI is an integrated AI development environment provided by AMD/Xilinx. It includes a DPU (Deep Learning Processor Unit), which performs convolution operations at high speed, and its associated interfaces, which are implemented as hardware in the PL (Programmable Logic) part of the Zynq device.

On the KV260, as shown in Figure 1, additional units are included to support DPU computation: one unit schedules instructions for the DPU, and another transfers data from external DRAM to BRAM, making the data accessible to the DPU. These units are connected to the CPU in the PS (Processing System) part via the AXI bus, enabling the DPU to perform operations on data fetched from DRAM.

Due to hardware limitations of the Zynq UltraScale+ on the KV260, only a single DPU is included in the basic design for this hackathon. However, for highly motivated participants, we also provide the option to use multiple DPUs with smaller hardware configurations.

It is handled by "xmutel" commands from the PS program.

B. *Implementation Procedure of YOLO v3*

In this hackathon, we used an image recognition program based on YOLO (You Only Look Once) v3 [5] as the application. The implementation on the FPGA is carried out following the steps shown in Figure 2. These steps are publicly available on [6], and students follow them as they proceed with the implementation.

The hackathon was conducted in sessions of 12 to 15 students, with each student provided with a KV260 board, cables, a web camera, and a pre-configured PC account.

C. *Training YOLO v3 Using Darknet*

After logging into the PC, students perform the training of YOLO v3 using Google Colaboratory and the darknet framework. For this purpose, a darknet folder is distributed to the students, which they first upload to Google Colaboratory. They then compile and execute the training by clicking through the code cells.

This step can take several hours and often runs into time limitations imposed by Google Colaboratory. In this hackathon, we distributed pre-trained weight data files. For students who wished to perform additional training, we provided access to a GPU-equipped server.

D. *Model Conversion*

The generated or distributed weight data files are then converted into a format compatible with FPGA.

The provided PCs are equipped with a Linux environment using WSL2. Students launch a terminal, select Ubuntu, and operate within this environment. A directory for Vitis-AI is preconfigured, into which students copy the trained weight data.

Next, they launch Docker for the Linux environment (Ubuntu on WSL(Windows Subsystem for Linux)). Within Docker, they activate conda, enable the vitis-ai-tensorflow2 environment, and use a script to convert the darknet model into one written with the Keras library.

With this setup, they can now run the quantization program. Using a pre-prepared Python script called my_quantizer.py, they perform quantization and then run the script compile_yolov3.sh to generate data files for the FPGA.

At this point, three files are generated: the model, weight data, and an md5sum.txt file. These files need to be transferred to the FPGA in order to run the application.

E. *Connecting KV260 board to the host PC*

Now, we connect the KV260 board to the host PC as shown in Figure 3 with the serial cable and the Ethernet.

To begin, a serial connection is established using a terminal program MobaXterm, and the power is turned on for the KV260 board. The board boots from an SSD card

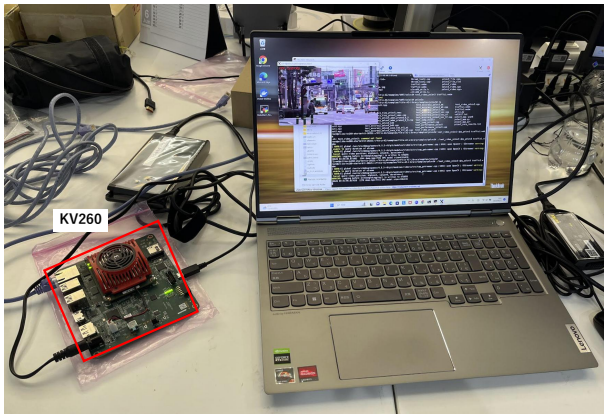

Fig. 3. Connection of the KV260 board

Fig. 4. A recognition example of a still picture

containing a Linux environment for experimentation, and Ubuntu Linux starts running on the PS (Processing System) of the KV260.

Next, a terminal is opened on the KV260 board, and the IP address is configured. Once both the Windows 11 settings on the PC and the IP configuration on the KV260 are completed, a local connection becomes possible. At this point, a second terminal is opened in MobaXterm, and SSH is used to access the board.

Once these steps are completed, the experimental environment is ready.

If students are comfortable with vim, they can edit files directly on the Linux system. However, since many students are unfamiliar with vim, VS Code is launched instead. VS Code is also useful later when working with images, so it's convenient to start it at this stage. Since it is pre-installed on the host PC, students only need to click the icon and select "Connect via SSH" to begin.

At this point, the following four windows are open:

- The Docker/conda environment window on the host PC (Docker window)
- The FPGA window opened in VS Code (VS Code window)
- The MobaXterm window used for serial connection (MobaXterm window)
- The FPGA window connected via SSH (FPGA window)

It is easy to get confused about which window to use, but the main ones are the Docker window and the FPGA window. The VS Code window is used only in special cases, and the MobaXterm window is only needed when rebooting or reconnecting the FPGA.

F. *Inference on FPGA*

In the FPGA window, create a working directory. Then, in the Docker window, transfer the three files generated during quantization — xmodel, md5sum.txt, and meta.json — to that directory using the scp command.

Next, in the FPGA window, copy the pre-prepared prototext file yolov3_coco_41_tf2.prototxt into the working directory. In this file, the num_class field should reflect the maximum number of labels for classification. It is set to 80 by default, so it should be changed to 3. This can be done either using vim in the FPGA window or with the code command in the VS Code window.

Once the setup is complete, by running a script file, two C++ files, test_jpeg_yolov3.cpp and test_file_yolov3.cpp, are compiled.

Next, load the design data into the PS (Processing System) and PL (Programmable Logic) parts. To do this, we use the xmutil commands. First, display the currently loaded application (xmutil listapps), then unload it (xmutil unloadapp), and finally load the newly compiled design data (xmutil loadapp kv260-benchmark-b4096).

As the test image, a still picture of cars is selected and copied from the Coco-image set, and the image recognition is executed with the compiled C++ code.

The result shown in Figure 4 is obtained.

G. *Video Recognition*

To perform video recognition, it is necessary to compile the program by linking it with Xilinx's VART (Vitis-AI Runtime Library). We used a script file to compile the C++ source file with the link of YOLO v3 model.

Executing the script generates a file named 'yolov3', and by entering the following command in the VSCode (FPGA) window, video recognition becomes possible. An example is shown in Figure 5. The FPS (Frames Per Second) is displayed in the top left corner and serves as a performance metric.

On the other hand, accuracy is evaluated using a graph of mAP (mean Average Precision) as shown in Figure 6. Students are evaluated with the FPS and the mAP.

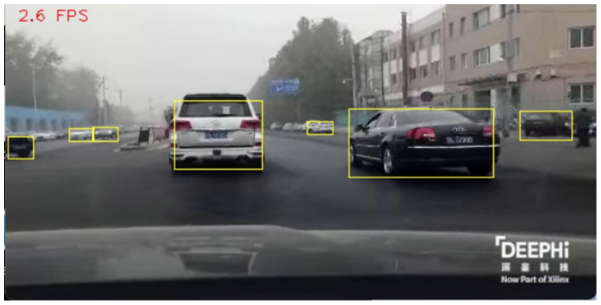

Fig. 5. An example of video recognition

```
cd ~/Vitis-AI/examples/Vitis-AI-Library/samples/yolov3
./test_file_yolov3 dpu_yolov3-608 val2017.txt
cd mAP
python3 main.py -na
```

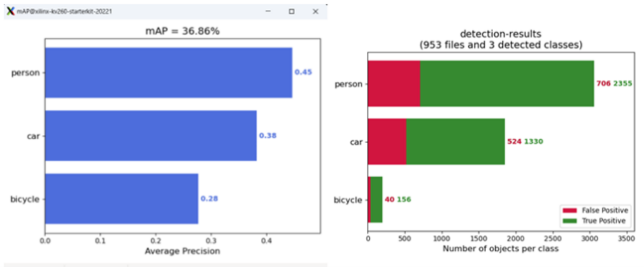

Fig. 6. mean Average Precision

H. *Optimization Strategy*

In the following steps, participants are free to modify the program and execution environment as they wish to improve performance and accuracy. However, we provide the following strategies as hints:

- Implement YOLOv3-tiny: Requires rebuilding the model
- Pipelining and multithreading: Sample code is provided for multithreading only
- Change the input image size: Sacrifices accuracy but improves performance
- Convert the initially used Keras model to an ONNX model
- Split the model using the ONNX model
- Tune the DPU

The program includes commented-out code to measure execution time. We encourage participants to first enable this code, measure the execution time of each process, perform profiling, and then proceed with optimization.

TABLE I
PARTICIPANTS OF 2024 HACKATHON

Under graduate Course	20	Graduate School	27
Machanical Eng.	6	Electrical	16
Electrical/Informatics	3	System Creation	2
Informatics	2	Material	1
Precision Eng.	2	Interdisciplinary	1
System Creation	2	Precision Eng.	1
Material	1	Social infra.	1
Aerospace	1	Compr. Culture	1
Liberal Arts	2	New Area	2
Literature	1	Electro. Inf.	1
		Intelligence Inf.	1

The final method involves modifying the use of the DPUCZDX8G integrated on the KV260 board, making it a highly advanced approach. The DPUCZDX8G allows the construction of one or multiple DPU cores of eight different sizes. The KV260 used here is equipped with only a single B4096 core and operates with the application called kv260-benchmark-b4096. Due to resource limitations, the KV260 can only accommodate one B4096 core.

Alternatively, it is possible to use smaller-sized cores and increase processing speed through parallelization by incorporating multiple cores—for example, using configurations such as B512×3–4, B1600×2, or B1024×3. However, whether sufficient resources are available depends on careful adjustment of the configuration settings. Another possible optimization approach is to increase the clock frequency. Participants who attempted this level of optimization used PCs with Vitis/Vivado installed.

IV. EXPERIMENTAL EXAMPLES

In the 2024 academic year, the Semiconductor Design Hackathon was held as an intensive summer course, open to both graduate and undergraduate students. Due to the limited number of participants per session, the hackathon was conducted over four days per session, with a total of four sessions and 47 participants overall.

To share results and present awards, a separate final presentation day was held, during which participants visited Hitachi's research laboratories for external presentations and discussions. During the hackathon, a lab tour of the Kosuge Laboratory was conducted, including microscope observations of actual chips, to spark participants' interest in the field.

The breakdown of participants is shown in Table I. There were 20 undergraduate and 27 graduate student participants, representing a diverse range of departments and majors. Notably, participants included students from humanities-related fields such as Liberal Arts and Liter-

TABLE II
Examples of Optimization

Design	FPS	mAP	Optimization Method
No Opt.	2.6	45	Using YOLO v3
Mitarai	75.0	24.3	Using YOLO v3-tiny Adjusting the timing of the displayFrame function call Reducing the frequency of waitKey Controlling thread execution order in parallel processing Changing image size
Pan	9 101	47 12	Using YOLO v3 Using YOLO v3-tiny Utilizing multiple DPUs via multithreading Trying new models such as YOLO v10-n Retraining according to Resize and Quantization
Yu	70.3	25.1	Using YOLO v3-tiny Increasing the number of threads Reducing display frequency

ature, which is particularly remarkable. All participants successfully carried out some form of optimization and were able to improve both performance and accuracy.

A. Examples of optimizations

Here, as examples of optimization by participants, the results of three individuals listed as authors of this paper are shown in Table II.

Many students attempted to switch to YOLO v3-tiny, a method that improves performance at the cost of accuracy. Mitarai enhanced performance by reducing the frequency of the displayFrame function execution. He modified the program to utilize an input buffer, allowing display only when the buffer became saturated.

The idea behind reducing the execution of waitKey—which detects key presses to terminate the YOLO process—is to slightly improve execution time by skipping some of these checks. Changing the image size is also effective for improving performance; however, this compromises accuracy, and retraining is necessary to maintain it.

Pan experimented with various models, retraining each to preserve accuracy. Furthermore, he utilized four available DPUs efficiently with parallel threads while implementing mechanisms to maintain the correct output order despite parallel execution.

Similarly, Yu also used multithreading and improved performance by reducing the display frequency.

V. Conclusion

This paper reports on the results of a hackathon in which participants implemented image recognition using YOLO on an FPGA board and competed in terms of performance and recognition accuracy, even without prior knowledge of hardware design. We provided detailed support, such as offering access to GPU resources for training and preparing a Vitis/Vivado environment for those interested in optimizing the DPU. On the other hand, some students independently achieved advanced optimizations without relying heavily on teaching assistants, instead using ChatGPT or exploring resources and sample programs available online.

Notably, students from humanities backgrounds performed on par with their peers. While most participants joined primarily out of interest in AI, they ended up gaining curiosity and enthusiasm for FPGAs, hardware, and semiconductors. We believe this means the hackathon successfully met its objectives.

In the 2024 trial, participation was limited to the University of Tokyo, but in 2025, the event was open to the general public. Eighteen teams participated, and the event is scheduled to take place as a research seminar on June 25, 2025. Additionally, an initiative to conduct the hackathon for a larger number of participants using AWS F1/F2 instances has also begun. The results of these trials will be reported at a later time.

References

[1] A.Kosuge and et.al, "Agile-X: A Structured-ASIC Created with a Mask-less Lithography System Enabling Low-Cost and Agile Chip Fabrication," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 33-3, pp. 746–756, Mar. 2025.

[2] Ministry of Economy, Trade and Industry, "AI edge contest," accessed: 2024-12-12. [Online]. Available: https://www.meti.go.jp/policy/mono_info_service/joho/aiedge/index.html

[3] Intel, "InnovateFPGA design contest," accessed: 2024-12-12. [Online]. Available: https://www.intel.co.jp/content/www/jp/ja/products/docs/programmable/innovate-fpga-design-contest.htm

[4] AMD/Xilinx, "Kria KV260 Vision AI Starter Kit," accessed: 2024-12-12. [Online]. Available: https://www.amd.com/ja/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html

[5] Joseph Redmon, Ali Farhadi, "YOLOv3: An Incremental Improvement," *arXiv:1804.02767*, 2018.

[6] M.Nitami, T.Goto , "Semiconductor Design Hackathon 2024 wiki," accessed: 2024-12-12. [Online]. Available: https://github.com/takgto/utokyo-chipathon2023/wiki/