h²ECU: A High-Performance and Heterogeneous Electronic Control Unit for Automated Driving

Xiebing Wang
Technical University of Munich

Kai Huang and Long Chen
Sun Yat-sen University

Alois Knoll
Technical University of Munich

Massive amounts of multi-sensor information pose a huge computational challenge for the design of a real-time automated driving module. The proposed h²ECU is a high-performance and heterogeneous electronic control unit (ECU) architecture for automated driving. By incorporating multiple multiprocessor SoCs (MPSoCs), general-purpose GPUs (GPGPUs), and FPGAs into one module, it can provide sufficient computing power while maintaining high scalability. The feasibility of h²ECU is demonstrated on a modified COTS vehicle, associated with a customized evaluation board and two typical advanced driver-assistance system (ADAS) applications.

The electronic component count and associated wiring content within modern vehicles have sky-rocketed as the automobile continues its transformation into an electronic computing system. Nowadays, a typical car contains dozens of ECUs, while the premium ones could have more than 100 ECUs.[1,2] This rapidly growing number of ECUs per vehicle has caused a paradigm shift in information and communication technology (ICT) architectures by reversing the growth trend of dedicated ECUs for specific functions towards integrating more and more disparate functions into one or a few control units. This is the so-called ECU consolidation. The trend is adding not only more ECUs but also more computing power. Such consolidated ECUs must provide not only multiple functionalities but also significantly more performance in terms of computational resources.

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

One of the functionalities that demands the most computing power is the ADAS, which is an auxiliary but essential part for autonomous driving. This system requires more and more computational resources as massive amounts of multi-sensor information need to be aggregated to assist in-car and on-road safety control. In 2012, Continental rolled out a schedule for autonomous driving, sketching a roadmap for fully automatic driving at higher speeds and in complex driving situations to be ready for mass deployment by 2020 and 2025, respectively.[3] Therefore, an ideal ECU module that demonstrates high performance but consumes low energy is imperative for automotive computing.

Traditional ECUs are not able to meet the requirements of autonomous driving, and not merely due to the huge performance demand. Over the whole production period of a car model, not all of the originally chosen ECUs will remain available in the market. Some of them will no longer be produced and have to be replaced by newer counterparts due to discontinuation of an ECU's specific technology.[1] Particularly for a consolidated ECU, how to cope with this trend still remains a pendent issue. Additionally, over the past 30 years, the amount of software has evolved from nearly zero to tens of millions of lines of code (LOC). A current premium car, for instance, implements about 270 functions deployed over about 70 embedded platforms. Altogether, the software amounts to about 100 Mbytes of binary code.[1] The next generation of upper-class vehicles is expected to run up to 1 Gbyte of software. The problem here is how to design an ECU to seamlessly cope with such a combination of mix-critical software as processor consolidation is closely aligned with the trend towards mixed criticality systems in which safety, security, and real-time critical components must coexist with fewer critical components.

This article proposes h^{2}ECU—a high-performance and heterogeneous ECU architecture for future automated driving. h^{2}ECU is a modular and reconfigurable architecture in which accelerators can be flexibly embedded to achieve high computational power. Based on this architectural design, we implement a prototype evaluation board that incorporates state-of-art MPSoCs, GPGPUs, and FPGAs into a heterogeneous system. The board is designed in a modular manner, where multiple GPGPUs and FPGAs are connected through the PCI Express (PCIe) interface, depending on the required computational power. In this way, the next generation of GPGPUs and FPGAs can be used while the ECU architecture remains unchanged. To program the onboard software, we use open computing language (OpenCL) as the programming framework. Applications programmed with OpenCL can be seamlessly exploited on both GPGPU and FPGA devices. Finally, we deploy two frequently used ADAS applications to demonstrate the use of the board in the real world.

OVERVIEW

Most of the traditional ECU products are for commercial use, and, consequently, companies scarcely release details about the technical implementations. At present, the industry is pushing ahead to the development of ADAS ECUs for large-scale deployment. At the CES consumer electronics convention in 2016, Qualcomm released its prototype product based on Snapdragon SoC 820A for next-generation automotive applications.[4] Meanwhile, NXP unveiled the MPC577xK series microcontroller for ADAS and industrial radar applications.[5] At CES 2017, Intel announced its GO automotive 5G platform[6] that would incorporate Xeon Phi processors and Cyclone V SoC FPGA, while Nvidia continues its promotion of the Drive PX 2 platform[7] and DriveWorks software.

The general use of autonomous driving vehicles is still not yet mature, and the majority of work lies in academia. Earlier studies about on-vehicle autonomous driving modules mainly focus on the ADAS implementation (whether autonomous driving tasks can be fulfilled by virtue of COTS components). In this case, portability, thermal constraint, and power consumption issues are entirely overlooked. The well-known Google driverless car does not publicly reveal any information about its computing system. However, the predecessor of the Google car, the Stanford Junior,[8] which won second place at the DARPA Urban Challenge 2007, is embedded with two Intel quad-core workstations. The winner of the same race, the Carnegie Mellon University (CMU) Boss,[9] is equipped with ten 2.16-GHz Core2 Duo processors. As can be observed, such heavy-computing functional components will become standard equipment for future vehicles.

Autonomous driving components should and must cooperatively work with off-the-shelf automotive ECUs and microcontrollers.[10,11] It has been a consensus that the functionalities of conventional on-vehicle ECUs have to remain unchanged to guarantee the hard real-time control of the vehicle, while new modules should be introduced to handle computing-power demanding automatic driving workloads. From this point of view, due to the advantage of high performance and scalability, hardware accelerators such as GPGPUs and FPGAs tend to be more important for future autonomous driving systems, because this solution can meet both the portability and real-time requirements of the autonomous driving module.

Following the above-mentioned trend, this article proposes a new ECU architecture for future automated driving. By integrating multiple GPGPUs and FPGAs into an embedded platform, the execution of automated driving tasks is effectively accelerated, while the whole system can be reconfigured to tune the performance demand.

ARCHITECTURE DESIGN

In general, an autonomous driving module is used to execute ADAS applications and then translate the results to trigger the launch of lower-level hardware execution units. These applications have diverse functionalities that implicate different criticality levels, such as lane detection, pedestrian detection, vehicle identification, traffic-sign recognition, and so on.

Figure 1 reveals the sketch overview of the h^{2}ECU architecture. The system consists of a fixed module where fundamental runtime infrastructures (host scheduler, runtime environment, OS support, I/O management, and so on) are rooted, as well as a reconfigurable module that enables computational power tuning by utilizing flexible hardware accelerators. As an abstraction, each ADAS application is treated as a stand-alone task that would be executed on a certain reconfigurable processing core, subject to the task scheduling on the host. The host scheduler allocates computational tasks to appropriate processing units according to both the scheduling policy and the working status of the processors. The reconfigurable processors are potentially high-performance accelerators that contain hundreds of parallel cores. Normally, the host CPU behaves as the scheduler while the processors are GPGPUs, FPGAs, or CPUs, as well.

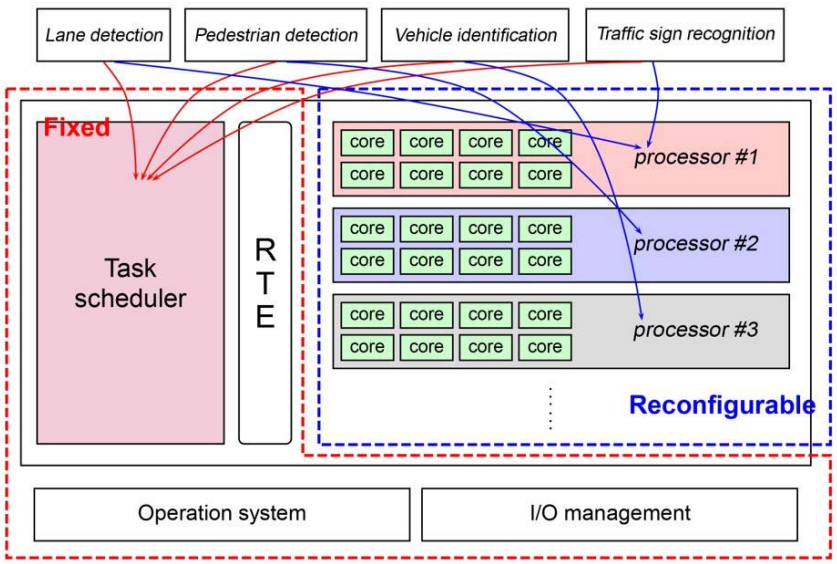

Figure 1. Sketch overview of h^{2}ECU. The components within the red dashed line are fixed, while the components within the blue dashed line are reconfigurable.

To coordinate the computing of these heterogeneous processors, we chose OpenCL as the runtime environment (RTE). By defining a high-level abstraction layer for low-level hardware

instructions, OpenCL enables cross-platform execution from general-purpose processors to massively parallel devices. In this way, autonomous driving tasks programmed with OpenCL can be seamlessly scaled and executed on a bundle of devices without any source code modification. Additionally, general-purpose operation system can be used to support the RTE and maintain the I/O communication of each component. With this design, adding new functionalities is cost-free, in terms of computing resources, as long as the RTE compatibility is met.

The benefit of this architectural design is multifold. Firstly, the spatial redundancy of the computing resources allows the coexistence of both safety- and non-safety-critical applications, thus providing appropriate partitioning mechanisms. Secondly, this modular design shows high flexibility, as multiple GPGPUs and FPGAs can be tuned depending on the required computing power. In this manner, next-generation GPGPUs and FPGAs can be used, while the ECU architecture can remain unchanged. Lastly, by virtue of high performance accelerators, the execution of automated driving tasks is effectively accelerated to guarantee the real-time constraint.

HARDWARE IMPLEMENTATION

We implemented the h^{2}ECU architecture on a prototype evaluation board. The board is a portable embedded platform where multiple MPSoCs, GPGPUs, and FPGAs can be assembled together. Figure 2 gives the top and side views of our third-generation prototype product. The board is a modular design so that as many components as possible can be reused in case of technology update and product upgrade. At runtime, the evaluation board is connected to another controller area network (CAN) bus communication board, which directly interacts with the auxiliary controllers within a vehicle.

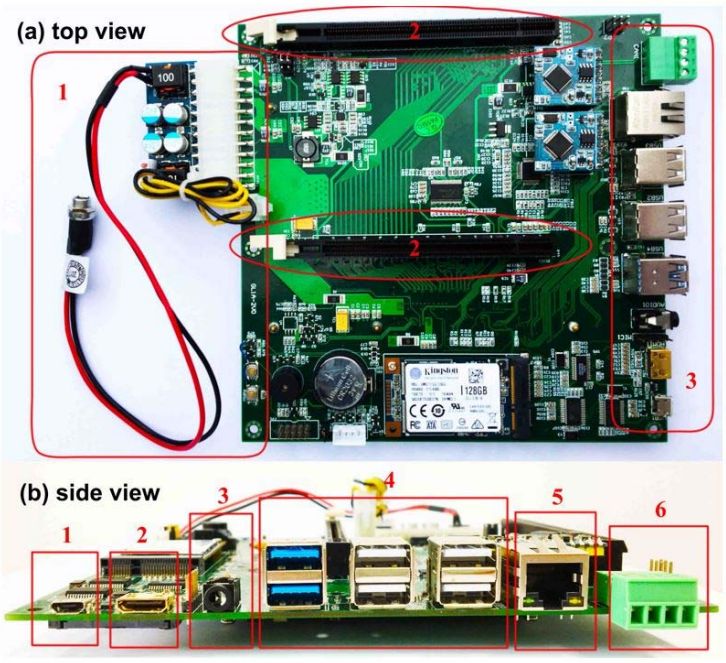

Figure 2. Hardware layout of the h^{2}ECU-based evaluation board. (a) Top view of the board: Part 1 is the power interface, Part 2 shows two PCIe slots, and Part 3 is the peripheral interface. (b) Side view of the board; the listed peripheral interfaces are (1) mini-HDMI, (2) HDMI, (3) audio port, (4) USB 3.0 and 2.0 ports, (5) Ethernet port, and (6) CAN-bus interface.

Our prototype product has five main components:

- *Power supply*. The power interface consists of two TPS54386 dual 3-A non-synchronous converters to ensure the stable and adequate dual power supply.

- *Host scheduler*. In the current generation, the board employs an Intel Core™ i5-3360M processor to schedule the ADAS applications. Although this power consumption is not suitable for practical use, we believe it is sufficient for prototype development.
- *Processing unit*. The h²ECU architecture supports a vast number of PCIe-based accelerators, and, at present, we use several low-energy-cost GPGPUs and FPGAs to handle the parallel computing tasks.
- *RTE*. To aggregate different devices, an installable client driver (ICD) loader is constructed and acts as a proxy between the user program and the actual implementations. In this way, OpenCL implementations of different vendors can be smoothly invoked without any conflicts.
- *Data communication*. The standard end-to-end PCIe data transfer protocol is used to facilitate the I/O interaction between the host CPU and computation devices.

Considering the component placement and space limit, the evaluation board consists of a 165-mm×165-mm mainboard, a thermal module, and extra PCIe accelerators. The mainboard provides a power interface, which consumes external power supply, and several peripheral ports used to connect external sensors. In the current generation, the mainboard contains two 16× PCIe slots, which support major state-of-art COTS accelerators. The platform is equipped with a 128-Gbyte SSD used for disk storage and an 8-Gbyte DDR3 RAM used for internal storage. The host processor is located on the other side of the mainboard (not visible in Figure 2), due to the need for heat dissipation. The heat of the CPU is both actively and passively dissipated by the thermal module, which is made up of an aluminum heat sink and a cooling fan (not shown in Figure 2).

ON-VEHICLE CONNECTION

We refitted the COTS sport utility vehicle DFM AX7 to support our use of the evaluation board. Figure 3 gives the logic layout of this testbed vehicle. The testbed contains mainly four layers:

- The upper layer consists of a series of external sensors, such as ultrasound sensor, lidar sensor, stereo camera, and so on. In this layer, the sensors capture the environmental information around the vehicle for further processing.
- The h²ECU-based platform is used to handle ADAS tasks in real time and gives response signals to the next layer.
- The middle layer includes an array of auxiliary controllers aiming at the engine management system (EMS), electric power steering (EPS), electronic speed control (ESC), and so on. These sub-controllers receive the command signals from the h²ECU-based platform and transmit them to the microcontrollers and hardware components.
- The lower layer is made up of (1) microcontrollers that correspond to the sub-controllers in the middle layer and (2) lower-level execution units such as engine, steering motor, brake solenoid valve, and so on. This layer controls the actual driving of the vehicle.

In the upper layer, the sensors acquire the environmental information and then transmit the data to the h²ECU-based platform. The ECU performs a series of ADAS algorithms and outputs the result, in form of command signals, to the middle layer. Afterwards, each auxiliary controller invokes a specific execution unit in the lower layer to control the driving of the vehicle. For instance, in Figure 3, the EMS controller receives throttle commands from the EMS sub-control and then regulates the engine. Then, the consequent torque and position signals generated by the steering wheel and pedal are received by the EPS and ESC controller, respectively. Subsequently, the EPS and ESC controller drive the steering motor and brake solenoid valve, together with the steering and braking commands from the EPS and ESC sub-controllers. The connection protocols between the upper sensors and the h²ECU-based platform are miscellaneous, while the communication between the ECU and the auxiliary controllers is through CAN bus.

EXPERIMENTS

We integrated several low-end GPGPUs and FPGAs into the evaluation board to ensure sufficient computing performance with rather low energy cost. In particular, we used Nvidia Quadro K600, Nvidia Quadro K620, and Nallatech PCIe-385N FPGA as test PCIe accelerators. To

demonstrate the proposed h²ECU architecture and the use of our evaluation board, we employed two commonly used ADAS applications on our prototype product.

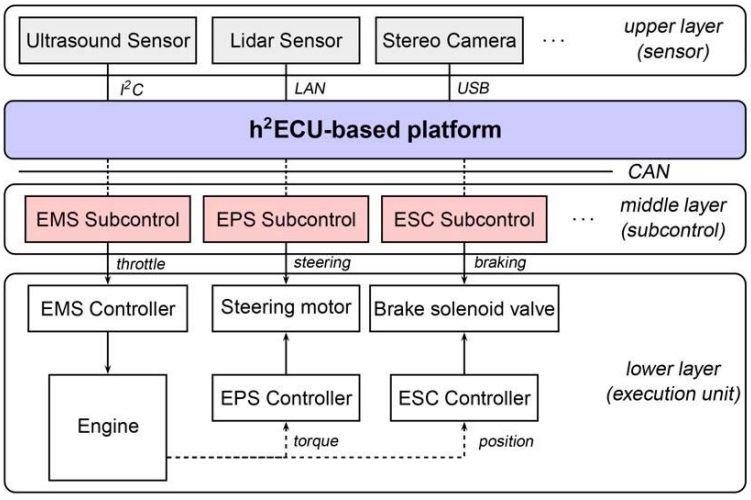

Figure 3. Abstraction of the layout of the testbed vehicle. The upper layer counts as the input for the h²ECU-based platform. The ECU platform performs data processing and generates outputs to the middle layer, which drives the concrete execution units in the lower layer.

Road lane detection (RLD)

This application is used to detect the lane markings while the car is driving on the road. The algorithm processes a video stream captured by a camera on a moving vehicle and highlights the positions of the lanes in the output stream. First, the image frames are pre-processed and transformed into gray-scale format so that the pixel intensity can be calculated. Then, by calculating the pixel weights of the lines from a randomly sampled line set, each of the lane markers is detected by selecting the line with the highest pixel weight. Afterwards, the lanes are tracked by applying a particle filter over the candidate lines from the previous frame, to predict the positions of lanes in the current frame. See our previous report[12] for more details about this algorithm.

In this application, the lane markings in each frame are either detected by the pixel weight ranking of the candidate lines or tracked by virtue of the particle filter, which predicts the positions of the lanes based on the position information of the previous detected frame. This functionality is implemented by an OpenCL kernel that can be computed across the different accelerators.

A series of videos recorded in different scenarios is used to reveal real-life road conditions. Table 1 gives detailed information about the test videos. These videos represent various road situations, including in day and night, with heavy traffic, with blurred and broken lanes, on the highway, and in urban and rural areas. To demonstrate the modular and heterogeneous features of h²ECU, we conducted the experiment with different configurations where either GPGPU, FPGA, or both are used to consume the kernel task. For the GPGPU-FPGA heterogeneous executions, we used a dynamic workload balance policy[13] to automatically accelerate the computation.

Traffic-sign recognition (TSR)

This application is used to detect and recognize the traffic signs that appear in the images recorded through the in-car camera. For the detection, based on the Haar-like features extracted from the image, a stage classifier is established and used to judge whether the content of a scaled scanning window within the image is a traffic sign. By performing a series of stage classifiers at different hierarchies, an AdaBoost cascade classifier is subsequently generated to collect the classification results of each stage classifier. In this case, the target image window is deemed a

traffic sign only if all the stage classifiers give a positive value. As for the traffic-sign recognition, first, a Gabor filter is adopted to extract features from the detected image windows. Then, these Gabor features are rarefied through principle component analysis (PCA) of the feature dimensions. Finally, the traffic signs are distinguished by means of linear discrimination analysis and template matching.

Table 1. Detailed information about the test videos for RLD.

Video	Resolution	Total Frames	Scenario
1	480×320	2,287	Broken lane
2	480×360	4,601	Crossing lane
3	640×360	899	Darkness
4	640×360	1,289	Rural area
5	640×480	232	Blurred lane
6	640×480	250	Bus view
7	640×480	337	Street shade
8	640×480	406	Blurred lane
9	640×480	1,718	Highway
10	640×480	1,897	Broken lane
11	640×480	2,654	Heavy traffic
12	640×480	2,799	Night highway
13	640×480	3,056	Street road
14	640×480	4,944	Light disturbance
15	640×480	4,992	Night
16	1,920×1,080	1,871	Highway

Haar-like feature extraction is observed as a performance bottleneck that can be parallelized to decrease execution latency. To demonstrate the high performance of our h^{2}ECU-based platform, we designed an OpenCL kernel for Haar-like feature object detection. As a comparison, a normal implementation of traffic-sign recognition through Haar-like feature extraction is customized as the baseline and another implementation using OpenCV API function cvHaarDetectObjects is proposed to showcase the speedup.

For this algorithm, the GPGPU is used as the accelerator, because the OpenCL SDK for Altera FPGA v13.1 does not support images. We test the algorithm with images under different resolutions, ranging from 160×120 to 1,920×1,080.

Performance results

We evaluate the performance of the RLD application in terms of input video resolutions. Figure 4 summarizes the performance results. As shown, in all resolutions, the executions on the evaluation board can achieve real-time performance. Even for high-definition 1,920×1,080 videos, the worst-case performance is observed as 37.7275 fps, when the tasks are consumed by the single PCIe-385N FPGA card. The results reveal that our h^{2}ECU-based platform is able to handle the RLD application in real time in a robust way.

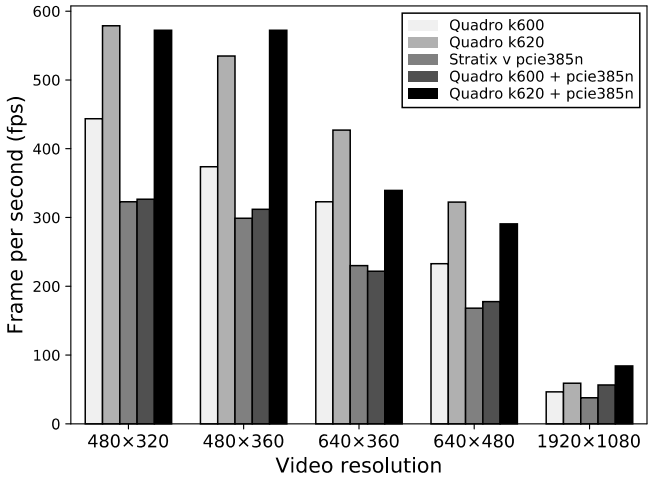

Figure 4. Performance of the RLD application in different video resolutions.

Another interesting phenomenon shown in Figure 4 is the heterogeneity of h^{2}ECU. That is, the best performance does not always lie on the single-accelerator execution. The Quadro K620 GPU outperforms all other configurations in most cases when processing videos with 480×320, 640×360, and 640×480 resolutions. However, the heterogeneous execution (Quadro K620 + PCIe-385N FPGA) turns out to be the best solution when processing 1,920×1,080 videos. This is extremely important, as state-of-art ADAS requires 1,920×1,080 definition. This means that future ADAS applications would favor reconfigurable architecture, such as h^{2}ECU, when the single homogeneous processing core cannot address the performance demand, not to mention the power and energy constraint.

Table 2 gives the execution time of our OpenCL-version TSR application on the evaluation board. All the executions can be completed in seconds, for different image resolutions. Although this application cannot fulfill the real-time requirement, the results are justified because the procedure and data manipulations of TSR are far more complex than RLD. Consequently, the computation task load of this algorithm is far larger than RLD, because a total of 14 stage classifiers are pipelined to do the calculation.

Table 2. Execution time of the TSR application using OpenCL kernels.

Image Resolution	Execution Time (ms)	
	Quadro K600	Quadro K620
160×120	26.68965	34.29279
320×240	67.53467	60.93644
640×480	237.8679	168.7996
720×480	266.0748	186.1955
1,280×720	699.5103	420.9746
1,920×1,080	1,612.417	921.8685

To better illustrate the high performance of the h^{2}ECU-based platform, we compared the performance of both the OpenCV- and OpenCL-based implementation of the TSR algorithm over the customized baseline. Figure 5 gives the experimental result. From the figure, we can see that the speedup of the OpenCV implementation over the baseline is within 3×, which is rather stable

across all image resolutions. However, the OpenCL implementation accelerates the application to a much larger extent, and this speedup ratio becomes greater when the image resolution increases. Specifically for 1,920×1,080 images, the algorithm can gain 11.38× and 19.90× speedup on Quadro K600 and K620, respectively. This reveals a huge potentiality for utilizing the h^{2}ECU-based platform to accelerate future ADAS algorithms.

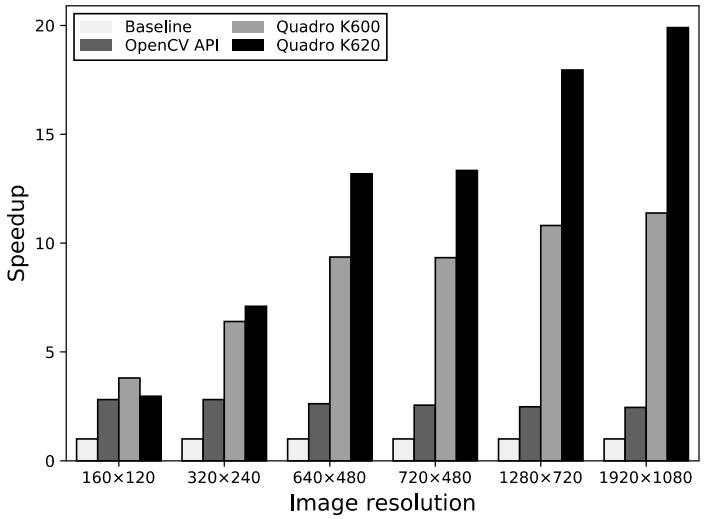

Figure 5. Speedup of the TSR application over customized baseline in different video resolutions.

DISCUSSION

Conventional ECUs for ADAS usually incorporate integrated SoCs to deal with real-time workloads. This is, however, hard to scale across different platforms due to technique upgrade and the consequent tedious software maintenance. By using reconfigurable architecture, the proposed h^{2}ECU in this article is able to leverage between the tradeoffs of performance demand and scalability design. Although it is hard to use state-of-art ADAS ECUs to test the pros and cons of our platform, the experimental study depicted in this article shines light on the future ADAS blueprint. The RLD application shows the performance benefit of heterogeneous architecture over its state-of-art homogeneous counterpart, while the TSR application reveals the huge potentiality to accelerate automated driving tasks through COTS accelerators.

The current generation of the evaluation board is not yet mature and still has some drawbacks. The biggest issue lies in the power consumption of the platform. In the future, we would like to adopt more energy-saving techniques and components to address the power constraint.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this article. This work is supported in part by a scholarship from the China Scholarship Council (CSC) under Grant Number 201506270152 and funding from the Science and Technology Innovation Committee of Shenzhen Municipality under Grant Number J20180170.

REFERENCES

1. M. Broy et al., “Engineering Automotive Software,” *Proceedings of the IEEE*, vol. 95, no. 2, 2007, pp. 356–373.

2. J. Cook et al., "Control, Computing, and Communications: Technologies for the Twenty-First Century Model T," *Proceedings of the IEEE*, vol. 95, no. 2, 2007, pp. 334–355.
3. "Continental Strategy Focuses on Automated Driving," Continental, 2012; www.continental-corporation.com/en/press/press-releases/automated-driving-128072.
4. "Snapdragon 820 Automotive Platform," Qualcomm, 2016; www.qualcomm.com/products/snapdragon/processors/820-automotive.
5. "MPC577xK: Ultra-Reliable MPC577xK MCU for Automotive ADAS & Industrial Radar Applications," NXP, 2016; www.nxp.com/products/processors-andmicrocontrollers/ powerarchitecture-processors/mpc5xxx-55xx-32-bit-mcus/ultrareliablempc57xx- 32-bit-automotive-industrial-microcontrollers-mcus/ultrareliablempc577xk- mcu-for-automotive-adas-industrial-radarapplications:MPC577xK.
6. "Intel R GO TM Automotive Solutions," Intel, 2017; www.intel.com/content/www/us/en/automotive/go-automateddriving.html.
7. "NVIDIA DRIVE: Scalable AI Platform for Autonomous Driving," Nvidia, 2017; https://www.nvidia.com/en-us/self-driving-cars/drive-platform/.
8. M. Montemerlo et al., "Junior: The Stanford entry in the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, 2008, pp. 569–597.
9. C. Urmson et al., "Autonomous Driving in Urban Environments: Boss and the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, 2008, pp. 425–466.
10. K. Jo et al., "Development of Autonomous Car—Part II: A Case Study on the Implementation of an Autonomous Driving System Based on Distributed Architecture," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 8, 2015, pp. 5119–5132.
11. S. Kokogias et al., "Development of Platform-Independent System for Cooperative Automated Driving Evaluated in GCDC 2016," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 4, 2018, pp. 1277–1289.
12. K. Huang et al., "A scalable lane detection algorithm on COTSs with OpenCL," *Design, Automation, & Test in Europe Conference & Exhibition*, 2016, pp. 229–232.
13. X. Wang et al., "Exploring FPGA-GPU Heterogeneous Architecture for ADAS: Towards Performance and Energy," *International Conference on Algorithms and Architectures for Parallel Processing*, 2017, pp. 33–48.

ABOUT THE AUTHORS

Xiebing Wang is a PhD student in the Institute of Robotics and Embedded Systems in the Department of Informatics at the Technical University of Munich. His research interests include autonomous riving, parallel computing, and heterogeneous computing. Contact him at wangxie@in.tum.de.

Kai Huang is a professor in the School of Data and Computer Science at Sun Yat-sen University. He has a PhD from ETH Zurich. His research interests include the analysis, design, and optimization of embedded systems, particularly in the automotive domain. Contact him at huangk36@mail.sysu.edu.cn.

Long Chen is an associate professor at the School of Data and Computer Science at Sun Yat-sen University. He has a PhD from Wuhan University. His research interests include autonomous vehicle perception system, computer vision and machine learning, point clouds processing, stereo vision, and driver behavior analysis. Contact him at chenl46@mail.sysu.edu.cn.

Alois Knoll is a professor of computer science in the Department of Informatics at the Technical University of Munich. He has a PhD from the Technical University of Berlin. His research interests include robotics, multi-agent systems, data fusion, adaptive systems, multimedia information retrieval, model-driven development of embedded systems, and simulation systems for robotics and traffic. Contact him at knoll@in.tum.de.