A Many-Ported and Shared Memory Architecture for High-Performance ADAS SoCs

Hao Luan and Yu Yao  
Horizon Robotics  
Shanghai 201210, China

Chang Huang  
Horizon Robotics  
Beijing 100094, China

Autonomous driving systems have attracted significant interest recently. Many existing industry leaders, such as Tesla [2], Nvidia [5], Mobileye, and Huawei [4], and a few startups, such as Horizon Robotics, have invested a large amount of capital and engineering power in developing advanced driver assistance system (ADAS) SoCs.

There are six levels of automation defined by SAE International [12]. Levels 1 and 2 of automation are mostly driving assistance, where the human driver still handles a substantial portion of the driving tasks under all conditions. Autonomous driving systems can take full driving responsibility at levels 3–5 of automation under certain driving conditions, which are typically referred to as highly autonomous vehicles (HAVs). Because HAVs represent the future of autonomous driving systems, we focus on HAVs at levels 3–5 for the rest of this article.

A typical SoC service for HAVs consists of the following processing pipelines and it should be able to complete it with current traffic conditions with a latency of 100 ms at a frequency of at least once every 100 ms or less [1].

1. **Object identification and tracking**: The video captured by cameras or other formats of raw data, such as data sourced by light detection and ranging (LIDAR) and radar [3], is streamed into both the object detection engine to detect objects and the localization engine to locate the vehicle in parallel. This step of processing is often accomplished with various machine-learning (ML) algorithms assisted by CPUs and other types of acceleration such as image cropping and region of interest (ROI) identification. Massive data are collected in real-time and a large number of training data sets (thousands of images) are also needed to infer and differentiate between vehicles and common objects such as traffic signs, pedestrians, and streets.

Digital Object Identifier 10.1109/MDAT.2022.3202997
Date of publication: 30 August 2022; date of current version: 24 October 2022.
2. **Fusion**: The vehicle location and tracked objects are projected into the same 3-D-coordinate space by the fusion engine, which is a processing step that combines the prior results into an understandable result. Again, a lot of heterogeneous processing occurs in this step, and there is a lot of data sharing between this step and the previous one.

3. **Motion planner**: In this step, the processed and analyzed data is now consumed by the motion planner to make operational decisions. The vehicle’s moving track and projected paths are calculated, and information such as navigation data is correlated. Heavy computations with a smaller and condensed data set are the characteristic of this processing step compared to the other steps.

4. **Control and decision**: The last and most important step in the entire flow. This step normally has the most stringent safety requirements to make sure the action and moving path are thoroughly checked and verified with the redundant pool of computing resources. Reliability is the key characteristic; however, it has the least amount of data to handle and process.

To fulfill such a real-time pipeline, one can easily observe that abundant ML and other heterogeneous processing power are undoubtedly needed on silicon. An efficient, domain-specific shared memory architecture with the following characteristics is also critical to keep all of them realizing at high tera operations per second (TOPS) computing capacity.

- **Big in size**: DaDianNao [9], an ML supercomputer, employs 36 MB of on-chip shared memory to accelerate ML applications. Tesla’s full self-driving (FSD) SoC uses two instances of a neural-network accelerator (NNA), where each NNA has 32 MB of on-chip SRAM memory providing a high-throughput data feed to enable and sustain a total of 72 TOPS processing power [2]. Huawei Ascend 901 SoC has two instances of 16 MB on-chip shared memory to enable a 256 TOPS processing power [4]. Therefore, a size of 32 MB of shared memory can be assumed as the baseline to accelerate heterogeneous ADAS processing with a lot of ML involved.

- **It is a many-ported and parallel architecture**: The data is shared among many relevant heterogeneous processing elements. It is always being accessed in parallel to provide high data throughput to all of them.

- **Provides consistent access latency optimized for the HAVs payloads [11]**: The raw image data, LiDAR, radar data, and the model data for ML processing are always available in the range of KB or MB. Thus, the architecture needs to fully utilize the bulky access nature of image processing and neural networks from a buffer level access as opposed to a single random byte or word access.

- **Provides necessary isolation**: Special care must be taken to provide sufficient isolation among the multiple parallel data paths to comply with ISO 26262 [7] requirements.

- **Software-friendly**: It should not behave as yet another conventional discrete memory on an SoC, where different pieces of the memory manifest different access latency and QoS. It needs to present to the software programmer as a flat and uniform memory space with consistent access latency across the entire memory space. This requirement can significantly ease the efforts of software programming and code management.

- **Scalable**: The same architecture can easily migrate from generation to generation from both logical and physical design perspectives and across multiple silicon manufacturing nodes.

### Related work and architecture

#### Related work

SoC architectures based on shared memory are the preferred backbone for flexible and programmable solutions in many application domains. Many-ported shared memory architecture has been explored recently both in academic and industrial settings.

Mesh-of-Trees (MoT) topology has shown that it can provide a consistent throughput as high as 98% for up to 64 masters and memory modules [10]. MoT topology consists of two phases: 1) the routing phase and 2) the arbitration phase. In the routing phase, every level of fan-out routing trees dilutes the traffic conflicts by half. This splitting process repeats as many as \( \log_2 N \) hops, where \( N \) is the number of masters. Hereafter, the data paths are merged by two gradually in the arbitration phase, which is also repeated in \( \log_2 N \) hops and the total data points...
converge back to \( N \) number of memory modules. The separation and isolation are well maintained in the routing phase. The splitting process is also an effective scheme to mediate memory access contentions. However, the isolation among multiple accessing masters, which is highly desirable in HAVs, is gradually lost in the arbitration phase. Besides, the architecture dictates a flat structure. This makes it harder to support modularity and scalability, which is very much required in engineering practice.

DaDianNao [9], which is an ML supercomputer, uses a Fat Tree topology consisting of a total of 16 data tiles to provide a 36-MB shared memory at a 28-nm technology node. It is big in capacity; however, the wires of the interconnect occupy half of the die area of each tile due to congestion. The two-level hierarchical structure is good to scale up the design but the global resource sharing has an intrinsic non-uniform memory access (NUMA) nature, depending on whether the producer and consumer of the data are on the same level or not.

Kalray MPPA-256 [8], a many-core processor targets real-time and embedded applications including ADAS. It has a total of 16 homogenous clusters, and each cluster has a 2-MB shared memory, which is shared by 17 identical very long instruction word (VLIW) cores without cache coherency. The shared memory comprises 16 independent memory banks. The memory banks are arranged on two sides (left and right). The PEs are organized into eight pairs. Each pair has two memory buses (one for each side of the memory group), which can be utilized in parallel by the two cores. As illustrated in [8], the interconnect topology employs two levels of crossbar switches where the first arbitration is done between the two cores in a pair and then the next level of arbitration is done among all pairs. The topology reduces a full crossbar of \( 17 \times 8 \) to two separate full crossbars of \( 8 \times 16 \) to two separate full crossbars of \( 17 \times 8 \). It is effective to reduce the overall number of wires, arbitration costs, and implementation complexity and to enable a shared memory of 2 MB with uniform memory access using a TSMC 28-nm HP process. However, the multilevel crossbars and round-Robin arbitrations really throttle the overall throughput, and the software/system engineers need to figure out how to avoid memory access conflicts among all the cores. Second, the homogeneity of the PE and its physical shape enables a perfect layout where all the PEs are sandwiched between the memory banks located on both sides [8]. This is something rarely available on an SoC that consists of many heterogeneous PEs along the data path.

A distributed and modular architecture is presented and studied in [6] to mainly optimize the real-time payloads for wireless applications while it tries to reduce the interconnect area at the same time. The architecture supports 32 masters with a size of 16 MB of shared memory, which is implemented on a 16-nm technology node. The architecture employs multilevel low-index switches to further reduce the wire complexity and arbitration costs that natively exist in many-ported interconnect and manage to confine the interconnect area to less than 30% of the total area. The hierarchical and modular approach is effective to make the overall architecture more scalable and easier to implement. The distributed approach can help to mediate memory access contentions and is also effective to mediate the NUMA effects. However, the divide-and-conquer approach taken to optimize interconnect area cannot satisfy the isolation requirement since it has merged the traffic from multiple parallel accessing masters right at the beginning of the data path.

Architectural challenges and considerations

Based on the analyses of the available architectures, the interconnect and its topology is the key factor that drives overall performance for a many-ported shared memory with a big capacity. To meet the ADAS requirements outlined in the previous section, important parameters such as how to connect many master ports to the sea of the memory instances, how to route, and to arbitrate the traffic generated by the parallel accessing masters need to be considered. Here is a summary of the challenges.

1. **Size matters:** A shared memory with a size of 32 MB or above normally consists of at least half a thousand to multiple thousand pieces of physical memory instances depending on the density and isolation requirements. The physical layout is very memory dominant and the NUMA effect is even worse with the growth of the size and the number of memory instances. Therefore, the physical implementation difficulties should be an important factor to consider.

2. **Mediation of the memory access contention:** Sharing by nature brings in access contentions among multiple parallel accesses generated by many heterogeneous processing elements. The interconnect, memory grouping, and memory
address mapping all need to be considered together to reach an efficient and cost-effective architecture.

3. **Isolation and sharing are a pair of contradicting requirements**: The contents of the memory need to be shared as much as possible to keep it local and reduce the power of data movement. However, the accessing data paths from the accessing masters also need to be kept isolated as much as possible to avoid unnecessary interference, which may be required after the ASIL allocation pursuant to ISO 26262 [7].

Domain-specific memory architecture

Figure 1 shows the overall architecture. It is a distributed and hierarchical one, which is applicable from both logical and physical design perspectives. A symmetric logical and physical partition is employed to increase the scalability and ease of implementation. The interconnect between the masters and the sea of memory instances is comprehended by a multiple-level recursive splitting and distributing structure. Just for simplicity, a two-level split and dispatching structure is shown to explain the concepts of the architecture. The bigger the size of shared memory, the more levels can be used to increase the parallelism. Therefore, less memory access contents can be achieved along with more parallelism enabled by this multilevel splitting process.

The splitting and dispatching are carried out in a recursive manner. Split by 4, 8, or even 16 can be considered based on the shortest burst size among the most frequently used burst sizes. For example, if bursts 4, 8, and 16 are the frequently used burst sizes

![Figure 1. Architectural view of the proposed shared memory controller.](image-url)
on an SoC, then a split-by-four structure is recommended to be used. A recursive split-by-four architecture is shown in Figure 2 to explain the splitting and dispatching scheme due to the popularity of this combination on today’s SoCs.

The following summarizes the rules to split and distribute a multibeat read command and write data inside the proposed architecture.

1. Disassemble any multibeat read requests and write data and then spread them across four clusters once they enter the shared memory.

2. Further introduce the next level of randomization, so the multiple beats within linear access go to a different SRAM array, to make sure it lands in a different memory bank to avoid access conflict inside a cluster and an SRAM array.

For example, if a burst 4 read command or write data is issued to the shared memory, every cluster gets one of the write data or read command beat as indicated by 0, 1, 2, and 3 shown in a different color. If a burst 8 or 16 read command or write data is issued to the shared memory, each cluster gets two or four beats of the read command or write data as shown in Figure 2 and they are guaranteed to fall in different SRAM arrays by applying the rules above. Enough randomization and whitening effects are introduced by applying the rules; the NUMA effect can be mediated significantly by averaging the individual beat access within a burst to achieve a consistent access latency to all multibeat read and write accesses. The above mechanisms successfully address architectural challenge No. 1 and No. 2.

The microarchitecture of an SRAM array is shown in Figure 3. At the input of an SRAM array, all the single or multiboot burst transactions have been split into single beat transactions in the previous split and dispatching structure, matching the data width of the SRAM instances inside. The dispatching logic decodes and routes the beat transactions to K logic banks based on the programmed addressing scheme. The arrangement of the memory instances is the direct reflection of the address scheme, which is the area where the grouping of the sea of the memory can be organized in a way to address the multiple requirements of ADAS applications. Schemes such as interleaving or hashing are helpful to increase the access parallelism of the logic banks when multiple masters try to access the same address range at the same time. The memory instances are grouped in a 2-D manner with a goal to further increase the parallelism to reduce access collisions while maintaining access isolations to some extent. Within an SRAM logic bank, the memory instances are sliced by regions. The 2-D arrangement completes a linear memory space with many vertical layers where each layer is realized by the subbanks fall in the same region across multiple memory banks.

For each logic bank, depending on the function safety and isolation requirement, the SRAM instances are separated into different subbanks where each subbank has its independent arbitration logic. Together with each master’s independent data path before the arbitration, this architecture can
provide complete separation for two masters accessing different subbanks. Note that if there is not any isolation required among multiple regions inside a memory logic bank, one arbitration logic should be sufficient for the whole bank.

With the replication of the arbitration logic, the data path from one accessing master to a memory region can be totally isolated from that of another accessing master accessing a different memory region. This scheme addresses challenge No. 3 and provides the fundamental support to satisfy the ASIL-B requirement defined by ISO 26262 [7] along with other safety mechanisms such as error correcting code (ECC) and timeout on the command and data. One may achieve ASIL-D results by leveraging system and software algorithms to compare and validate two ASIL-B data from two independent sources together [7]. The number of the logic banks in an SRAM array, how many subbanks are in a logic bank, and the number of the accessing masters supported are cost-sensitive to the total area and power consumption of the architecture. Therefore, these three important parameters need to be examined properly to achieve a good function/cost ratio.

Implementation, results, and correlations with the architecture

Multiple configurations have been studied in this architectural exploration. One prototype with the following configurations has been validated end-to-end from RTL simulations all the way to the physical design signoff.

1. A shared memory with a size of 32 MB consists of over half a thousand instances of physical memory.
2. Sixteen accessing masters with 256 bit read and write data width employing AXI 5 protocol with read data chunking supported.
3. Two levels of splitting and distributing by four structures are employed to complete the interconnect of the shared memory.
   • Sixteen memory banks inside each SRAM array.
   • The shared memory has an outstanding capability of eight commands right at every master port interface and an extra buffer worth storing 64 splitting and dispatching beats.
   • The interconnect network, which is represented by the two levels of splitting and distributing structure, runs at 1 GHz.
   • All memory instances run at 500 MHz.

To correlate with the overall architecture shown in Figure 1, the above configuration can be summarized as: $X = 16$, $M = 4$, $N = 4$, and each cluster has 8 MB of physical memory.

Simulation results

To fully examine and validate the effectiveness of the proposed architecture, traffic is injected together using various numbers of parallel accessing ports combined with different traffic patterns per accessing the port. Each port is injected with 10,000 read or write transactions in the simulation window. The throughput per port, the average read and write access latency for burst transfer, and bulk data transfer performance are analyzed. Figure 4 shows the read and write performance with different numbers of parallel accessing masters ranging from 1 to 16.

Each of the accessing masters issues random read and write requests with a random address (256 bit aligned) at the same time with a 100% injection rate. Figure 4 clearly shows that even if the accessing master numbers vary from 1 to 16, the throughput per accessing port is stable for both the read and write transfers. The actual port throughput is around 96% for the read traffic and around 99% for the write traffic. The port throughput does drop along with the addition of the accessing masters. However, the proposed architecture demonstrates a strong resilience to sustain the heavier traffic and only drops about 0.01% point for the read and about 0.46% point for the write throughout the whole range. This result is equal to or better than that of MoT [10] topology but with a much coarser level of splitting and distributing structure, which directly translates to the cost saving of the implementation while keeping the scalability and modularity. The average read and write latency shows similar robustness in the same settings. Even though the maximum access latency degrades when more accessing masters are added, the average read latency stays almost the same, and the average write latency degrades just a few cycles. It validates that the multiple levels of splitting and distributing structure have fully randomized the read and write traffic, so the NUMA effect can be tamed properly to provide consistent QoS. The results are better than those of [6] with a much simpler architecture.

Similar results can be obtained if traffic consists of random burst 4 transactions, burst 8, and even combined traffic with three different burst lengths.
Only the results based on burst 16 traffic are shown just to avoid repeating similar results.

To better mimic the real traffic patterns on an ADAS SoC, the second simulation is created to test the performance of bulk data transfers. This simulation is done with 16 masters reading and writing bulk data at the same time from each accessing master in parallel. And all accessing memory spaces do not have any overlap to comply with the isolation requirement.

As shown in Figure 5, the “ideal” number is calculated with 100% data bus utilization. The number of cycles to transfer such an amount of data can be calculated as the size of the transfer payload divided by the size of the data width of the bus interface. For example, it takes \((4 \times 1024) / 32 = 128\) cycles to transfer 4 KB of data in an ideal condition. It clearly shows that after the initial datapath pipeline latency, which is 32 cycles for the read, is incurred in the first burst, there is not any more extra latency introduced. It proves that the actual data throughput is close to 100% for the bulk data read. The bulk write also has a similar result that close to a 100% utilization rate is achieved after the first write is completed. Just as shown in Table 1, a bigger average read latency is observed in Figure 4 because the number of outstanding commands is set to 16 per master port, which is slightly bigger than the maximum number of outstanding commands set for the burst 16 traffic by design, to achieve the highest throughput. However, the average read access latency is settled at 36 cycles once the number of read outstanding commands is reduced to one. The average latency of the writes is more consistent compared with that of reads because the write data is always current and the splitting buffer is deep enough to take in enough beats and remove bubbles between the multiple writes.

One of the heaviest traces is then picked to further stress the architecture. This is one of the traces obtained from our earlier version of SoC. In this trace, every master has a 2-MB memory space reserved for its read and write accesses. Each of the master 0–7 runs with memory traffic from an in-house single-shot detection network, the size of data ranges from less than 4 KB to around 260 KB; each of the master 8–15 reads and writes with ROIs based on a 1080p YUV422 image where each ROI clips at 2 MB if it is bigger. The setup is to mimic that eight PEs run in parallel to process the images from up to eight cameras and all the masters read and write the data as defined in the traces. Figure 6 shows that the read latencies fluctuate a bit more in ML features and weights than in the image data. This is because a different subregion of the feature might be read in a different layer’s...
consistent access latency and throughput can be achieved.

Physical design results

The architecture has gone through the entire physical design signoff flow using a TSMC 7-nm technology library. As shown in the purple region in Figure 8, the modularity and hierarchical concept enabled by the architecture make it very easy to realize such a big design using either Synopsys or Cadence EDA tools. The physical design can take a hierarchical approach that once one cluster is implemented as a hard macro, four clusters are ready to be integrated with proper tie-offs and can be timing closed and signed off in the next higher level even in parallel with a short turnaround time.

The NUMA-aware architecture enables a much smoother timing closure to otherwise challenging timing paths to memory macros. All the memory macros can run at half speed as that of the interconnect because the randomization of the data return can easily absorb a slightly longer timing path segment to the SRAM macros. This enables us to select standard Vt (SVT) SRAM macros with higher density and good aspect ratios. The relaxation of the timing on SRAM leads to an 8% dynamic power reduction and about 11% leakage reduction for the entire processing. The feature access pattern of repetitively accessing a portion of a line and then a jump to the next line leads to more bank access conflicts. The access pattern of the image data is to continuously access the full ROI one line after another. Also, shorter burst lengths such as burst 4 or 8 are used by the PEs instead of burst 16, which is consistently used by the data transfer of image data. These two contribute to the differences in average read latency between the two types of traffic. Other than the above, the overall throughput is still close to 100%, which is no different from randomly generated data simulations, and so are the latency and throughput for the writes.

More results obtained from traces with less access intensity are omitted since the overall results are slightly better with almost the same throughput as shown in Figures 6 and 7. The results correlate well with the architectural analysis that the longer the burst, the better randomization, the easier a

![Figure 5. Read and write latencies with different size of payloads.](image)

| Setting | Number of Read | Number of Outstanding/Port | Stable Read Latency | Average Read Latency |
|---------|----------------|---------------------------|---------------------|----------------------|
| 1       | 16             | 16                        | 2.2                 | 36                   |
| 2       | 16             | 1                         |                     |                      |

Table 1. Latency with different numbers of outstanding.
SRAM macros. It also leads to less congestion in the layout where one can only see a few and tiny red regions (which indicates high congested areas) in the timing closed layout shown in Figure 8.

The physical layout is similar to the paper layout shown in Figure 1 but all the I/Os from all accessing masters are split into two groups and arrive from the center openings on the east and west sides. The

Figure 6. Read performance with different ML and ADAS traces.

Figure 7. Write performance with different ML and ADAS traces.
architecture enables flexible assignment of the I/Os to comply with top-level SoC routing and connection needs. The overall utilization of the silicon area is improved because of the removal of I/Os on the north and south sides. One step further, the I/Os can be pushed further toward the center with two benefits: 1) shorten the timing paths from the I/Os to the memory interconnect and 2) it can also spare some areas at the center of the two edges to accommodate some top-level SoC routings. The two levels of splitting and merging structure strike a good balance of the performance and wire density for such a wire and memory dominant architecture, the total area comes slightly under 30 mm² with roughly less than 36% area for the interconnect logic. The utilization inside each cluster is around 40% and the utilization in the top level is around 35% for the non-SRAM macroarea. This is better than the result reported in [6], in which low radix switches are used for the interconnect to improve utilization.

**We present a** new shared memory architecture target to high-performance ADAS SoCs. The results show that it can provide close to 100% throughput in both read and write for ML accelerations and ADAS-related raw data processing. It realizes consistent QoS for the domain-specific payloads and it represents a big and flat memory space with necessary isolations complied with ISO 26262 [7], which significantly simplifies the software programming and code management effort. It has a native modular architecture; it can scale up and out with a much smoother and easier implementation flow as demonstrated in the previous section. All of the positive results lead to the successful adoption of this architecture into our production SoCs that target to HAVs.

Moving forward, we actively investigate to further improving the consistency of the access latency, leveraging the scalable architecture to support even more shared memory and master access ports with 3-D integration and Chiplet techniques.

**Acknowledgments**

We would like to thank the production team of Horizon Robotics to provide relevant data to compare and analyze the benefits of this work. Special thanks to Horizon Robotics’ SoC design, functional verification, and physical design teams. This work is impossible without their outstanding support in RTL coding, functional simulation, and physical design. This article was presented in 2022 and appears as part of the *IEEE Design&Test* Special Issue.

**References**

[1] S. Lin et al., “The architectural implications of autonomous driving: Constraints and acceleration,” in *Proc. 23rd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2018, pp. 751–766, doi: 10.1145/3173162.3173191.

[2] E. Talpes et al., “Compute solution for Tesla’s full self-driving computer,” *IEEE Micro*, vol. 40, no. 2, pp. 25–35, Mar./Apr. 2020, doi: 10.1109/MM.2020.2975764.

[3] V. K. Kukkala et al., “Advanced driver-assistance systems: A path toward autonomous vehicles,” *IEEE Consum. Electron. Mag.*, vol. 7, no. 5, pp. 18–25, Sep. 2018, doi: 10.1109/MCE.2018.2828440.

[4] H. Liao et al., “DaVinci: A scalable architecture for neural network computing,” in *Proc. IEEE Hot Chips 31 Symp. (HCS)*, Aug. 2019, pp. 1–44, doi: 10.1109/HOTCHIPS.2019.8875654.

[5] Nvidia. *NVIDIA Unveils NVIDIA DRIVE Atlan, an AI Data Center on Wheels for Next-Gen Autonomous Vehicles*. Accessed: Jul. 26, 2022. [Online]. Available: https://nvidianews.nvidia.com/news/nvidia-unveils-nvidia-drive-atlan-an-ai-data-center-on-wheels-fornext-gen-autonomous-vehicles
H. Luan and A. Gatherer, “Combinatorics and geometry for the many-ported, distributed and shared memory architecture,” in Proc. 14th IEEE/ACM Int. Symp. Netw.-on-Chip (NOCS), Sep. 2020, pp. 1–6, doi: 10.1109/NOCS50636.2020.9241708.

[7] ISO26262. Accessed: Jul. 26, 2022. [Online]. Available: https://www.iso.org/standard/68383.html

B. D. de Dinechin et al., “Time-critical computing on a single-chip massively parallel processor,” in Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE), 2014, pp. 1–6, doi: 10.7873/DATE.2014.110.

Y. Chen et al., “DaDianNao: A machine-learning supercomputer,” in Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit., Dec. 2014, pp. 609–622, doi: 10.1109/MICRO.2014.58.

A. O. Balkan, G. Qu, and U. Vishkin, “A mesh-of-trees interconnection network for single-chip parallel processing,” in Proc. IEEE 17th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP), Sep. 2006, pp. 73–80, doi: 10.1109/ASAP.2006.6.

[11] N. Capodieci et al., “Real-time requirements for ADAS platforms featuring shared memory hierarchies,” IEEE Design Test, vol. 39, no. 1, pp. 35–41, Feb. 2022, doi: 10.1109/MDAT.2020.3013828.

[12] SAE International. Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems. Accessed: Jul. 26, 2022. [Online]. Available: https://www.sae.org/standards/content/

Hao Luan is the chief SoC architect at Horizon Robotics, Shanghai, China. His research interests include domain-specific SoC architecture, multide SoC architecture, and dataflow-based scheduling. Luan received a master’s in electrical and computer engineering from the University of Alberta, Edmonton, AB, Canada, and a master’s in automation from Tsinghua University, Beijing, China. He is a Senior Member of IEEE.

Yu Yao is an ASIC design engineer with Horizon Robotics, Shanghai, China, focusing on memory hierarchy, system scheduling, and Datapath optimization. Yao received a master’s in electrical engineering from the University of California at Los Angeles, Los Angeles, CA, USA.

Chang Huang is the CTO and one of the co-founders of Horizon Robotics, Beijing, China. His interests focus on high performance end-to-end optimizations of ADAS system solutions, software and hardware partitions for ADAS applications, AI algorithms, and corresponding compiler optimizations. Huang received a PhD in computer science from Tsinghua University, Beijing.

Direct questions and comments about this article to Hao Luan, Horizon Robotics, Shanghai 201210, China; hao.luan@horizon.ai.