Apurv Ashish, Sohan Lal, Ben Juurlink

An Efficient Lightweight Framework for Porting Vision Algorithms on Embedded SoC

An Efficient Lightweight Framework for Porting Vision Algorithms on Embedded SoCs

Apurv Ashish, Sohan Lal, Ben Juurlink

Technische Universität Berlin, Germany
apurv.ashish@campus.tu-berlin.de
{sohan.lal, b.juurlink}@tu-berlin.de

Abstract. The recent advances in the field of embedded hardware and computer vision have made autonomous vehicles a tangible reality. The primary requirement of such an autonomous vehicle is an intelligent system that can process sensor inputs such as camera or lidar to have a perception of the surroundings. The vision algorithms are the core of a camera-based Advanced Driver Assistance Systems (ADAS). However, most of the available vision algorithms are x86 architecture based and hence, they cannot be directly ported to embedded platforms. Texas Instruments (TI) embedded platforms provide Block Accelerator Manager (BAM) framework for porting vision algorithms on embedded hardware. However, the BAM framework has notable drawbacks which result in higher stack usage, execution time and redundant code-base. We propose a novel lightweight framework for TI embedded platforms which addresses the current drawbacks of the BAM framework. We achieve an average reduction of 15.2% in execution time and 90% reduction in stack usage compared to the BAM framework.

Keywords: Embedded systems, computer vision, autonomous systems, ADAS, TI TDA2xx

1 Introduction

Advanced Driver Assistance Systems (ADAS) can fully or partially assist a human during the driving process. The rapid rise of ADAS in the automotive industry is a prime example of the transforming capabilities of embedded vision technology. As shown in Figure 1, the recent years have seen unprecedented growth in the field of embedded vision systems which has transformed it from being a research-oriented field to real industrial use-case. The significant increase in applications implementing embedded vision systems has been fuelled by the recent improvements in the area of embedded hardware [7], and availability of accurate and robust algorithms.

In principle, an ADAS can be implemented with different types of sensors such as radar, lidar or camera, however, considering the current trends, camera-based applications are gaining more popularity [2]. The compute-intensive algorithms needed to process the camera inputs require specifically designed embedded hardware for delivering real-time performance. However, these algorithms

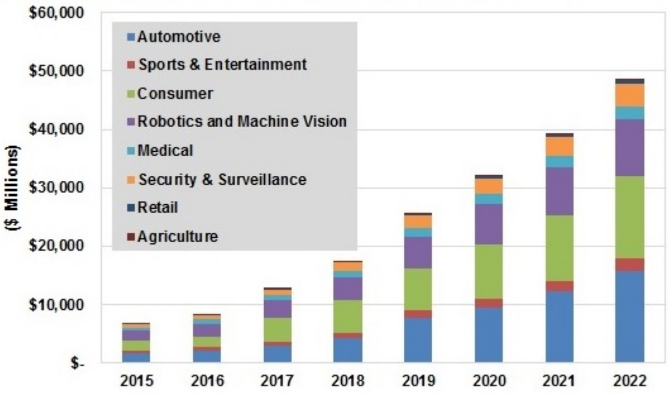


Fig. 1. Growth in embedded vision based applications [1]

cannot be ported on embedded platforms out of the box because most of the significant vision libraries such as OpenCV are x86 architecture based. Moreover, embedded platforms may use specific algorithm standards such as Texas Instruments XDAIS standard [17] and proprietary compilers which are not standard compliant as GCC compilers. Furthermore, the limited memory constraints and specialised SIMD requirements are other factors which are not addressed in x86 algorithm development. Therefore, the commonly available algorithms have to be adapted before they can be ported on an embedded platform.

The TI's TDA2xx [11] platform, which is also the target platform for evaluation in this paper, provides a Block Accelerator Manager (BAM) framework [16] for porting of x86 algorithms on embedded platforms. The BAM framework provides abstraction and improves programmability, however, the BAM framework also has a few drawbacks such as graph-based execution, more complexity (exposed interfaces) and limited DMA functionalities which influence the porting time of the algorithms as well as the runtime of a ported algorithm.

In this paper, we propose and develop a new lightweight framework aiming to rectify the drawbacks of the BAM framework. The term lightweight essentially refers to the conceptual gravity of the framework, i.e., the reduced number of exposed interfaces and the required code changes during the process of algorithm porting. The proposed framework significantly reduces execution time and stack usage of an ported application.

In summary, following are the main contributions of the paper.

- We propose a novel lightweight framework for porting computer vision algorithms on embedded platforms.
- We address several drawbacks of the BAM framework such as graph-based execution, best block search.
- Our framework provides an average 15.2% reduction in execution time and 90% reduction in stack usage compared to the existing BAM framework.

2 Related Work

There exists a myriad of approaches to port vision algorithms on embedded platforms [3–5, 8, 14, 18, 6]. These diverse approaches are primarily dictated by the spectrum of different available processing platforms and different programming models associated with these platforms. The use of ARM-based architectures allows the use of standard programming models, however, they are not well suited for compute-intensive applications. Hence, the use of Graphics Processing Units (GPUs) or vector processors is encouraged to accelerate compute-intensive applications to achieve higher energy efficiency. However, GPUs do not support real-time operating systems (RTOS) or similar frameworks which makes it challenging for scheduling of hard real-time applications such as ADAS [9].

An ADAS application consists of different algorithms as building blocks and performance optimization strategies for the individual algorithms have been thoroughly investigated [10, 12]. In [10], Nieto et al. propose a design and development methodology in the form of an iterative cycle which enables development of quick prototypes and further optimising and tuning specific aspects of the algorithms. Apart from the algorithm optimization, the "time to market" aspect has also been given fair attention. The difficulty of implementing an entire ADAS application on the hardware has led to hybrid solutions, where software and hardware implementations are combined to obtain the desired performance [14, 5, 18, 6].

In [8], Kocić et al. propose a methodology for porting of ADAS applications on heterogeneous platforms based on the computational requirement of different algorithms. They propose a method of decoupling an application such as a face-detection application into different parts depending on the type of computational requirements and algorithm-specific optimizations. The pixel manipulation part of the application, for example, colour space conversion is ported on SIMD cores (GPUs), whereas face-detection algorithms are implemented on DSP cores.

J. Perez proposes porting of the brake-by-wire application to an embedded platform using a directed acyclic graph (DAG) [3]. The DAG is used to present the schedule of the application threads to six processing elements and the dataflow between the application threads, enabling task and temporal parallelism. Gostovic et al. propose a similar approach for porting of vision algorithms on an embedded platform [4]. They port an object detection algorithm to TI TDA2xx platform using a DAG. The DAG is used to model the data-flow between different computation nodes as well as the input and output nodes. TI provides Block Accelerator Manager (BAM) framework to expedite the process of porting vision algorithms on its embedded platforms [16]. We observe that the BAM framework has several drawbacks such as graph-based execution which leads to higher stack usage.

In contrast to the above-mentioned works, we propose an efficient lightweight framework for porting of vision algorithms on embedded platforms that rectifies several drawbacks of the existing BAM framework.

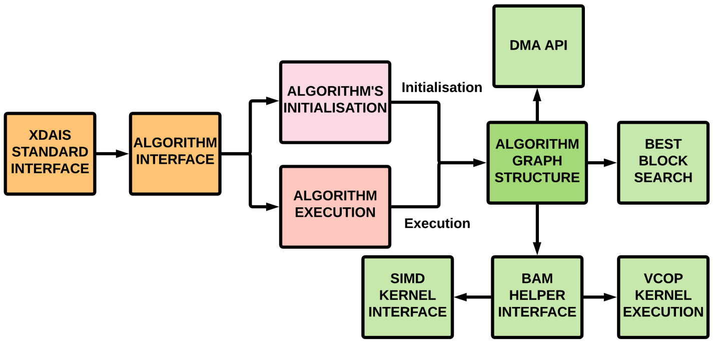


Fig. 2. Block diagram of BAM framework (adapted from [16])

3 Background

3.1 TDA2XX System-on-Chip

TI's TDA2xx [11] is a high-performance System-on-Chip (SoC) based on TI's open multimedia applications platform architecture. The architecture is specifically designed for front camera and surround view applications. The embedded vision engine [13] (EVE) subsystem in TDA2xx platform is a programmable vector processing engine, best suited for pixel manipulation operations such as color space conversion. TDA2xx platform supports four independent EVE subsystems.

EVE subsystem consists of an ARP32 scalar core, a vector coprocessor (VCOP) SIMD unit, and an Enhanced DMA (EDMA) controller. The ARP32 is a 32-bit scalar core and functions as the subsystem controller. The critical features of the scalar core include the control and coordination of EVE's internal interactions and interaction with other subsystems in the TDA2xx SoC. The VCOP is a SIMD engine with built-in loop control and address generation functionality. The EDMA block is the local DMA unit for the EVE subsystems. The EDMA unit is used for transferring data blocks between system memories (typically SDRAM and/or L3 SRAM) and internal EVE memories (data buffers).

3.2 Block Accelerator Manager Framwork

To expedite the process of porting algorithms on EVE subsystem, TI provides Block Accelerator Manager (BAM) framework. Figure 2 shows the block diagram of the BAM framework. The prime objective of the BAM framework is to simplify and speedup the porting of vision algorithms on TI's embedded platforms. The BAM framework achieves this objective by performing block-based image processing rather than processing full image frame in a single execution. The block-based processing is particularly suited to the needs of embedded platforms. Further, the BAM framework provides abstraction of the data movement and vector programming with the implementation of the directed acyclic graph.

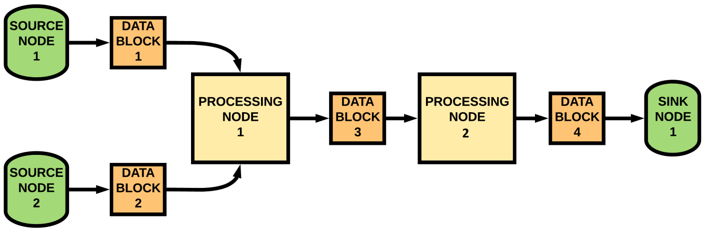


Fig. 3. Graph based execution (adapted from [16])

As shown in Figure 2, the entire framework is subdivided into two parts: XDAIS interface and algorithm interface. The XDAIS interface is designed to enable multiple algorithms to coexist and share system resources. The co-existence of various algorithms is ensured by preventing hard-coded use of critical system resources such as memory, DMA, and other accelerators. The XDAIS interface also enables a client application to query the algorithm specific requirements.

The algorithm-specific exposed interfaces are used by a client application for the execution of the algorithm. As shown in Figure 2, the algorithm initialization interface is used by the client interface for the initialization of an algorithm's graph structure. In the algorithm initialization, the reference to specific APIs or functions is assigned to different nodes of the graph as designed by a programmer. The execution interface is used by the client application to call the graph structure at the time of execution. The algorithm-specific computation code is executed on the vector co-processor which supports a C-variant programming model. The computation code is called through a wrapper rather than an explicit call to the algorithms computation function. As shown in Figure 2, the helper functions basically initialize and execute the computation code. The input and output DMA transfers are realized by the source and sink nodes of the directed acyclic graph.

As shown in Figure 3, the DAG represents the source/sink nodes for data input/output and processing nodes where the actual computation takes place. The source and sink nodes are programmed in standard C implementation which controls the TI's EDMA (Enhanced Direct Memory Access) unit. The processing nodes are implemented using Kernel-C, the TI's vector core programming language which handles actual algorithmic computation. Further details about the BAM framework can be found in [16].

4 Proposed Framework

Figure 4 shows the block diagram of the proposed framework called Efficient-BAM (E-BAM). The framework is designed to address drawbacks of the BAM framework as explained later. The proposed approach offers the same level of abstraction as provided by the BAM framework. We introduce substantial architectural changes to enhance the porting mechanism of vision algorithms on

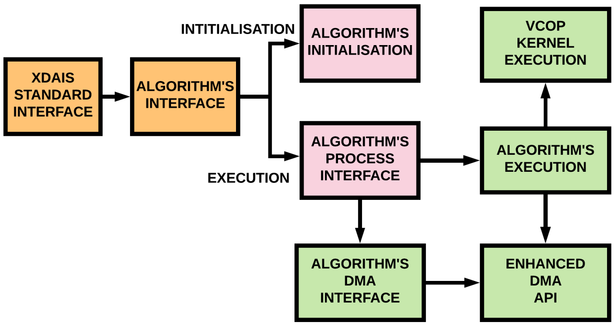


Fig. 4. Block diagram of the proposed framework

TDA2xx platform which results in improved runtime performance. In the following subsections, we describe the main architectural changes to the existing BAM framework as well as the motivation for these changes. The green blocks in Figure 4 highlights the major architectural changes in the proposed framework.

4.1 Modular Execution Model

The abstraction provided by the BAM is achieved primarily through a graph creation which results in higher stack memory usage because the graph structure is stored in the EVE on-chip memory during execution. Further, the graph object is referenced for every image block, which results in the call overhead. Hence, the graph creation adds to the overall execution time of an application.

In the proposed framework, instead of creating a wrapper such as a graph structure for the DMA operation, memory allocation and kernel execution, we perform these operations explicitly but in separate modules. This approach ensures lower stack usage, which is required to store the graph object and removes the latency due to repeated calls to the graph object. It also makes code more transparent and easy to extend due to its modular nature.

4.2 Enhanced DMA API

The block-based processing requires image blocks to be repeatedly fetched, processed and written back to the external memory. The BAM framework provides two basic functions, i.e., auto-increment and scatter-gather DMA. We extend DMA API with new features such as short inline functions, link transfer and chain transfer [15]. In general, an application may require repeated DMA transfers and the inline functions provide considerable improvement as it reduces the considerable function call overhead. The transfer use cases such as the transfer of input blocks between predefined memory location benefit from the link transfer feature because it removes the requirement of repeated parameters update. Similarly, the chain transfer feature automatically triggers the next transfer event after completion, hence, reducing the overhead of triggering a new transfer event.

Table 1. Qualitative comparison

Key Features	BAM	E-BAM
Graph creation	Online	No graph creation
Best block search	Runtime	Offline
DMA	Generic TI DMA library	Enhanced DMA API

4.3 Best Block Search

In block-based image processing, dimensions of a block can have a significant effect on the performance of the application. The BAM framework offers a block optimization method, which deduces the optimum block dimensions, keeping valid memory allocation during the process. However, the drawback with this approach is the expensive processor cycles spent in search of optimal block dimensions during run-time.

In the best block search algorithm for the E-BAM framework, we initialize the block width and height of an image segment with the values of 16 and 8 respectively. The choice of this value is influenced by the SIMD width of the vector processor. In order to find optimum dimensions, we iteratively increase block dimensions and calculate the required memory size. When the required memory size exceeds the available memory, the block dimensions in the previous iteration are accepted as optimum block dimensions. This process ensures that all the data fits in the available memory without leaking or overwriting each other. Since the memory size of the EVE subsystem and image size are known before hand, the offline computation of the best block is justified. Further, since the ping-pong buffers in the EVE subsystems have only one read and write port, we ensure that the input and output blocks are not written back to the same buffer, thereby eliminating the waiting time before the block is read or written back to the buffer.

Table 1 shows the summary of key features of BAM and E-BAM frameworks. In the BAM framework, graph initialization takes place during runtime of an application, however, in the E-BAM framework, the graph structure is not used. On contrary, the E-BAM framework uses a mix of modular and sequential programming to provide the same level of abstraction with improved runtime performance. The best block search feature required for deducing the optimum block dimensions is calculated during the execution of an application, however, in the E-BAM framework, the best block search is performed offline. The BAM framework uses the generic TI DMA library with basic functionalities, whereas the E-BAM framework is equipped with the customized DMA API which supports additional features such as short in-line functions, link and chain transfers.

5 Experimental Setup

To evaluate our proposed framework, we use TI TDA2xx as the test platform. For benchmarking of the baseline BAM and proposed framework (E-BAM), we choose image pyramid generation as the test application. The choice of the image pyramid generator for evaluation is justified considering its complexity, intensive computation and DMA requirements. We perform benchmarking for the BAM and E-BAM frameworks using two different test vectors at three different frequencies (535, 600 and 650 MHz) for the TDA2xx platform. Although, the number of cycle count provides an accurate measure of the execution time, we present our results for three different frequencies as it reflects the practical use case where the TDA2xx platform can be operated at three different frequencies. We present execution time which is the average of five consecutive test runs for each test vector. Moreover, the timing measurement for TDA2xx platform is done in total number of processor cycles which can account even small variations.

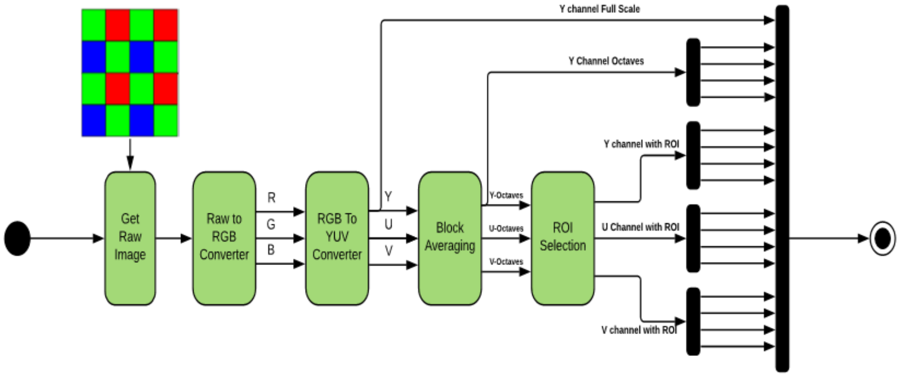


Fig. 5. Pyramid generation application

As shown in Figure 5, the pyramid generation from raw camera input comprises of three different algorithms, all executing in parallel with the output of the first algorithm (RAW to RGB conversion) serving as the input for the second algorithm (RGB to YUV conversion) and the output of the second algorithm as the input for the third algorithm (Block Average). The application also requires synchronization of data-block movement to/from the external memory. The image pyramid application is also suitable for measuring the DMA overhead in both frameworks since it requires intensive DMA operations due to 17 different output images to be written back to off-chip memory.

6 Experimental Results

We compare our efficient lightweight framework (E-BAM) with the existing BAM framework. We provide quantitative comparison of stack usage, execution time at algorithm level as well as at application level.

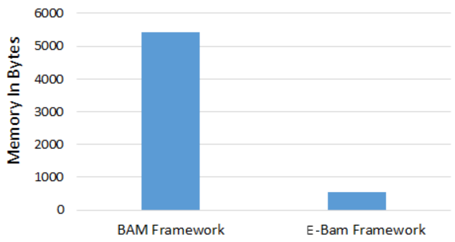


Fig. 6. Stack usage comparison

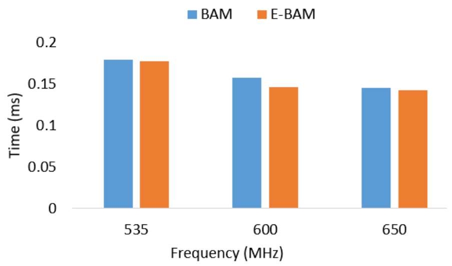
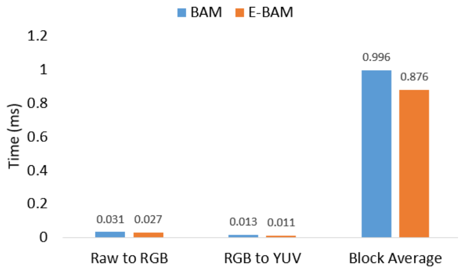


Fig. 7. DMA overhead



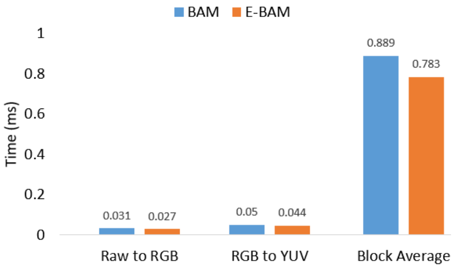


Fig. 8. Execution time for test vector 1 **Fig. 9.** Execution time for test vector 2

6.1 Stack Usage

Figure 6 shows the stack usage in bytes of E-BAM and BAM frameworks. We see that the proposed framework reduces 90% stack usage compared to the BAM framework. The significant reduction in the stack usage is the result of not using graph-based execution. The reduction in stack usage is very important as it opens up possibilities for processing of larger block dimensions.

6.2 DMA Overhead

The DMA overhead is the time which processor spends while waiting for completion of input or output block transfer. Figure 7 shows the proposed framework reduces the DMA transfer time compared to the BAM framework. The improvement is largely due to the improved DMA API with short inline function calls and additional features such as link transfer that we implemented for customized DMA transfers.

6.3 Execution Time at Algorithm Level

Figure 8 and Figure 9 show the execution time of individual algorithms for BAM and E-BAM frameworks at 600 MHz. We see that all three algorithms (Raw to RGB, RGB to YUV and Block Average) achieve significant reduction

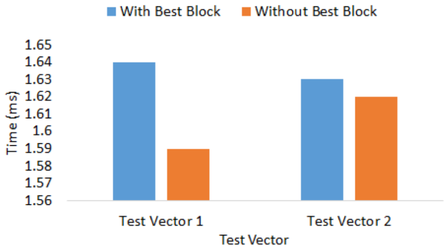


Fig. 10. Overhead of best block search

in execution time compared to the BAM framework. The average reduction in execution time for the three algorithms is approximately 12%. The considerable reduction in execution time is due to the reduced complexity in the handling of kernel computation code in the E-BAM framework. The BAM framework involves function calls to BAM helper functions which act as an interface between the kernel computation code and the executed function. However, in the E-BAM, we initialize and call the kernel computation function explicitly.

6.4 Best Block Search Overhead

As shown in Figure 10, the best block search algorithm incurs a significant amount of overhead in the BAM framework. The results shown in Figure 10 are the average of five consecutive runs for the same algorithm on two different test vectors with platform running at 600 MHz. The execution without the best block search is performed with the block dimensions of 256x16 (width x height), which is calculated manually. The best block search increases the overall execution time. The best block search can also lead to variation in the execution time due to different amount of time incurred to search the best block, which can be problematic to model in a time-budget based hard real-time system. The E-BAM algorithm does not suffer from the overhead and execution time variation since the best block is computed offline.

6.5 Execution Time at Application Level

Figure 11 and Figure 12 show the execution time of the entire application (including the initialization, algorithmic computation and DMA transfer) for BAM and E-BAM frameworks at three different frequencies. Again, we see that the proposed framework reduces the execution time for different frequencies. The average reduction in execution time is about 15.5% and 14.9% for the first and second test vector, respectively. On average the proposed framework delivers an average reduction in execution time of 15.2% compared to the BAM framework. In addition to the reduced complexity in the handling of kernel computation code and enhanced DMA API, the removal of additional overhead due to repeated calls to graph object also contributes to the reduced execution time of the E-BAM framework.

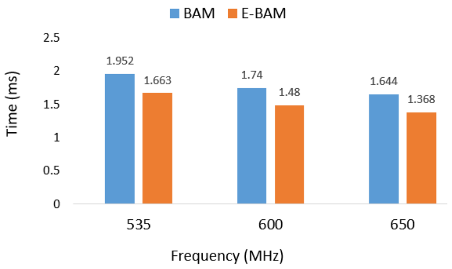


Fig. 11. Execution time for test vector 1

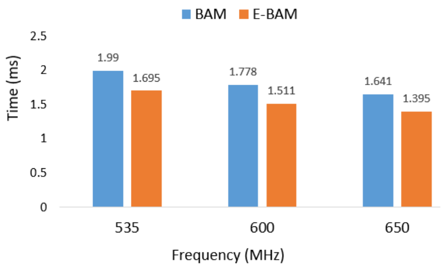


Fig. 12. Execution time for test vector 2

7 Conclusions

The recent improvements in the performance of embedded hardware and accuracy of artificial vision systems have made autonomous vehicles a reality. However, different hardware architectures, programming models and varying use cases make porting and optimization of vision algorithms on embedded platforms an uphill task. The BAM framework provides a significant improvement in terms of programmability and abstraction for porting of vision algorithms on TI platforms, however, it has several drawbacks. We propose a novel lightweight framework E-BAM for porting of vision algorithms on TI platforms. We optimize the BAM framework by removing graph-based execution, shifting the best block search to offline and customizing the DMA API which supports additional features such as link transfers. Our proposed framework reduces average execution time by 15.2% along with 90% reduction in the stack usage compared to the existing BAM framework.

References

1. Computer Vision Hardware and Software Market to Reach $48.6 Billion by 2022 (2016), https://www.tractica.com/newsroom/press-releases/computer-vision-hardware-and-software-market-to-reach-48-6-billion-by-2022/
2. Global Automotive Camera-based ADAS Market 2018-2022 (2018), https://www.researchandmarkets.com/reports/4516968/global-automotive-camera-based-adas-market-2018
3. Gonzales-Conde Perez, J.L.: Analysis of Task Scheduling for Multi-Core Embedded Systems (2013)
4. Gostovic, M., Pranjkic, M., Kocic, O., Maruna, T.: Experience from Porting Complex Algorithms on Heterogeneous Multi-Core Systems. In: IEEE 7th International Conference on Consumer Electronics-Berlin (ICCE-Berlin). pp. 235–238 (2017)
5. Hsiao, P.Y., Yeh, C.W.: A Portable Real-time Lane Departure Warning System Based on Embedded Calculating Technique. In: IEEE 63rd Vehicular Technology Conference, 2006, VTC. vol. 6, pp. 2982–2986 (2006)
6. Jeng, M.J., Guo, C.Y., Shiau, B.C., Chang, L.B., Hsiao, P.Y.: Lane Detection System Based on Software and Hardware Codesign. In: 4th International Conference on Autonomous Robots and Agents, ICARA. pp. 319–323. IEEE (2009)

7. Kisacanin, B., Bhattacharyya, S.S., Chai, S.: Embedded Computer Vision. Springer Science & Business Media (2008)
8. Kocić, O., Simić, A., Bjelica, M.Z., Maruna, T.: Optimization of Driver Monitoring ADAS Algorithm for Heterogeneous Platform. In: 24th Telecommunications Forum (TELFOR). pp. 1–4. IEEE
9. Maghazeh, A., Bordoloi, U.D., Eles, P., Peng, Z.: General Purpose Computing on Low-Power Embedded GPUs: Has it Come of Age? (2013)
10. Nieto, M., Vélez, G., Otaegui, O., Gaines, S., Van Cutsem, G.: Optimising Computer Vision Based ADAS: Vehicle Detection Case Study. IET Intelligent Transport Systems **10**(3), 157–164 (2016)
11. Nikolic, Z., Agarwal, G., Williams, B., Pearson, S.: Ti Gives Sight to Vision-enabled Automotive Technologies. Texas Instruments, Technical Report (2013)
12. Poudel, P., Shirvaikar, M.: Optimization of Computer Vision Algorithms for Real Time Platforms. In: 42nd Southeastern Symposium on System Theory (SSST). pp. 51–55. IEEE (2010)
13. Sankaran, J., Hung, C.Y., Kisačanin, B.: EVE: A Flexible SIMD Coprocessor for Embedded Vision Applications. Journal of Signal Processing Systems **75**(2), 95–107 (2014)
14. Stein, G.P., Rushinek, E., Hayun, G., Shashua, A.: A Computer Vision System on a Chip: A Case Study From The Automotive Domain. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops, CVPR. pp. 130–130 (2005)
15. TI: KeyStone Architecture Enhanced Direct Memory Access (EDMA3) Controller (2015), http://www.ti.com/lit/ug/sprugs5b/sprugs5b.pdf
16. TI: BAM Algorithm Framework User Guide (2019), http://www.ti.com/tool/processor-sdk-tdax, linux and RTOS Processor SDK for Vision V3.06.00
17. Torud, S.: A Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers (2002), http://www.ti.com/lit/an/spra579c/spra579c.pdf
18. Velez, G., Cortés, A., Nieto, M., Vélez, I., Otaegui, O.: A Reconfigurable Embedded Vision System for Advanced Driver Assistance. Journal of Real-Time Image Processing **10**(4), 725–739 (2015)