A Computer Vision System on a Chip: a case study from the automotive domain

Gideon P. Stein Elchanan Rushinek Gaby Hayun Amnon Shashua
Mobileye Vision Technologies Ltd. Hebrew University
Jerusalem, Israel Jerusalem,Israel
Stein,Rushinek,Hayun@mobileye.com shashua@cs.huji.ac.il

Abstract

The automotive market puts strict and often conflicting requirements on computer vision systems. On the one hand the algorithms require considerable computing power to work reliably in real-time and under a wide range of lighting conditions. On the other hand, the cost must be kept low, the package size must be small and the power consumption must be low. In addition, automotive qualified parts must be used both to withstand the harsh operating environment and to guarantee long product life.

To meet all these conflicting requirements Mobileye developed the EyeQ, a complete 'system on a chip' (SoC) which has the computing power to support a variety of applications such as lane, vehicle and pedestrian detection. This paper describes the process of designing an ASIC to support a family of vision algorithms.

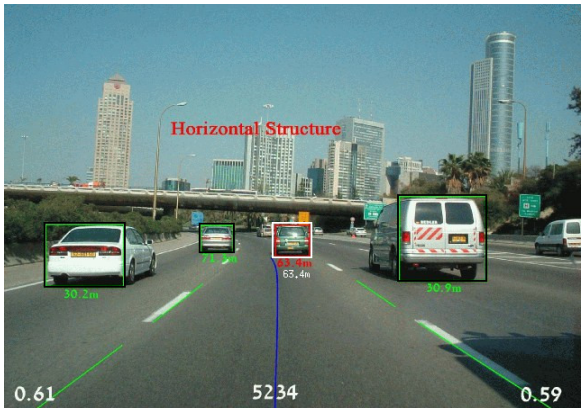

Figure 1. Typical automotive vision application: detection of lanes, vehicles and other scene elements such as bridges from a single forward looking camera.

1 Introduction

There is a strong and growing automotive market for computer vision systems. A large number of automotive applications require the ability to locate the road structures including boundaries and lane markings and to locate objects such as vehicles and pedestrians. Figure 1 shows the results derived from a single forward-looking camera mounted on a vehicle. This information can then be used for Lane Departure Warning, Forward Collision Warning, Lane Change Assistance (with the camera mounted on the side mirror) and Collision Mitigation by Active Braking. Pedestrian Protection (figure 2) is another up and coming automotive application.

The applications are required to work reliably in a large range of lighting and climactic conditions and to work in real-time at frame rates of 15-30 FPS or even higher. The algorithms can initially be developed on powerful desktop computers or powerful embedded processor platforms. However for production they must be ported to a low cost, compact computing platform.

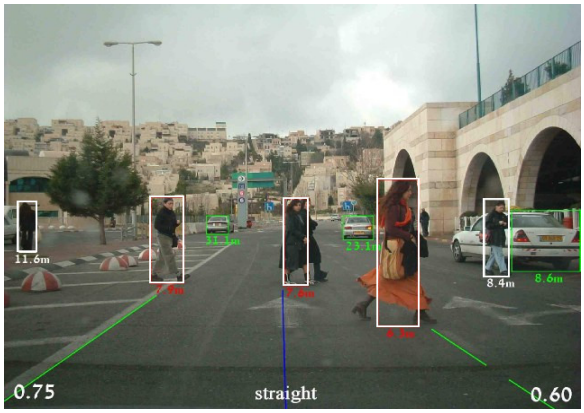

Figure 2. Detecting pedestrians, a highly variable class of non-rigid objects, is a task demanding an exceptional amount of computation.

Figure 3. The EyeQ chip packaged in an automotive qualified 292 pin Heat Sink Ball Grid Array (HSBGA).

To meet this challenge Mobileye developed the EyeQ, a System-on-a-Chip (SoC). The chip (figure 3) was designed from the start for the automotive market. Running at under 2W in power dissipation the chip can support applications that would require a large desktop computer. Furthermore, its high level of integration enables very compact designs. For example, the SeeQ unit combines an EyeQ processor, image sensor, memory and power supply on a $53mm \times 33mm$ board (see figure 4).

Section 2 gives an overview of the EyeQ chip architecture. The main focus of the paper is section 3 which describes the design process of the EyeQ chip. The steps that follow the hardware design such as manufacturing are not discussed.

2 Overview of the EyeQ chip architecture

The EyeQ chip block diagram is shown in figure 5. The chip is composed of two ARM processors (CPU) and four specialized vision computing engines (VCE). One ARM CPU is used for chip control, communication to the vehicle and general IO. The second ARM CPU is used for implementing algorithms more suitable for a general purpose processor rather than specific hardware.

The main computational power of the chip comes from its four VCE units:

1. Classifier Engine:
    - Image scaling and preprocessing.
    - Image pattern classifier
2. Tracker Engine:
    - Image warping
    - Image tracking
3. Lane Detection Engine:
    - Lane marks detection
    - Road Geometry detection
4. Window, Preprocessing and Filter (WPF) Engine:
    - Image convolver
    - Image pyramid creation
    - Edge detection
    - Image filters

Figure 4. The SeeQ board measures only 53mm $\times$ 33mm and is an example of a compact embedded application which includes processor, HDRC image sensor, memory and power supply.

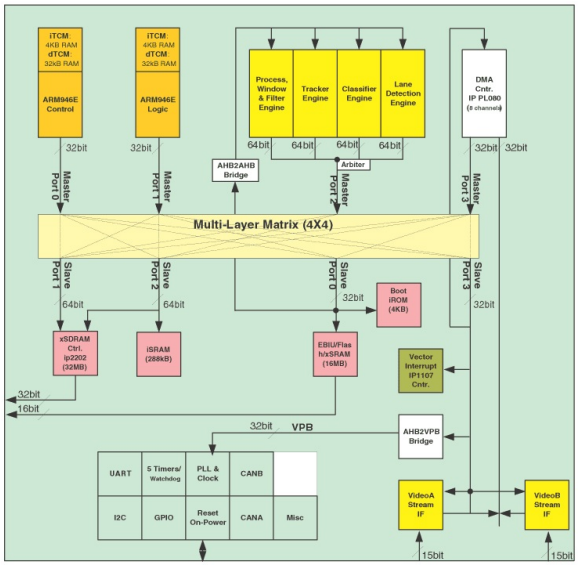

Figure 5. The EyeQ chip block diagram. The key elements are the two ARM processors (top left), the four Vision Engines (top center), the DMA controller (top right) and the video port (bottom right) all connected to the internal and external memory through a 4x4 multi-layer matrix for high bandwidth parallel IO.

- Image histogram

The EyeQ supports two 12-bit, 640x480 resolution video streams. It provides a 12-bit glue-less interface to a variety of image sensors including high dynamic range CMOS (HDRC). There is hardware support color space conversion, Bayer de-mosaicing and 12-bit to 8-bit pixel compression.

3 EyeQ design process

This section describes the 'front end' design process which started from the basic system definition and ended with a logic level description of the chip. This process was roughly composed of the following steps:

1. Defining the system requirements: (1) Target applications and algorithms (2) Video and communication IO requirements.

   The EyeQ chip was designed to run Mobileye's algorithms for lane detection, vehicle detection and pedestrian detection[3]. These algorithms are the core of the application. A general purpose CPU must be on chip to provide the application wrapper. This CPU is also used to tailor the application for each particular customer.

   The basic IO requirements were two video channels to allow for multiple cameras, CAN 2.0b ports for communication with the vehicle, a RS232 serial port and some general purpose IO pins to make a standalone application. The video channels were designed to have glue-less interface with a variety of BW and color CMOS sensors.

2. Defining the hardware and software architecture: This is a very important step which includes analysis of the algorithms to determine which tasks require hardware implementation and then choosing an appropriate hardware architecture to optimally support the target algorithms. These tasks were grouped together and became the VCEs. The main hardware tasks are: video input, basic image processing, road and lane analysis, pattern classification and object tracking. Section 3.1 describes this analysis in more detail.

3. Initial VCE design: In this step it was determined what sort of data paths and memory structures would be used for the hardware implementation of each block. As described in section 3.1 each block has unique requirements. One output of the initial hardware design step is a set of equations which give the cycle count and memory bus requirements for a given task. Some tasks have fixed execution time. For others, the execution time depends on task parameters which are data dependent. For example, tracking depends on the size of the object to be tracked.

4. Algorithm simulation to estimate run time using VCEs and fine tuning hardware resources: By running the algorithm on a desktop computer using real image sequences, one can generate a trace of all the tasks together with the parameters used. Combining this with the equations of the VCE performance, one can determine the run time of the algorithm on the proposed hardware (assuming serial execution).

   By defining dependencies between tasks, the trace output from the serial desktop computer can also be used to simulate parallel execution where we assume a simple scheduler which keeps to the same order of execution. While not an optimum scheduler, it does give a worst-case estimate of the computation time for each frame. Using this information one can see the bottlenecks in the hardware design and increase hardware resources such as arithmetic units or registers.

   This step also highlights the tradeoffs between memory bus bandwidth and extra computation. In general, computation is cheap in terms of area while local memory can become quite expensive. The memory bus is

also a valuable resource not to be wasted. For example, in the case of the tracking unit it was better to compute the spatial derivatives on the fly at each iteration rather than storing the results locally or bringing them in from the derivative image computed by the preprocessing unit.

5. Define detailed specifications of VCEs including data and task control interfaces: In this step the tasks and instruction sets for each VCE are define. This includes the bit structure of each task command, how data and parameters are packed to be sent to the VCE, and how results are returned. As part of this step software emulators were written for testing.

6. Design of test vectors to test each VCE individually.

7. Fixed point analysis: The basic algorithms on the serial computer were written using a mixture of fixed point and floating point. An important design decision was to determine whether floating point was required on the EyeQ chip. To help in this decision, a tool was developed so that for each variable it would generate a histogram of the values it held. This was done even for temporary variables which are not explicitly named in the program code. Using this tool one can determine where fixed point can be used (and what precision is required) and where floating point (hardware or emulation) must be used. The results showed that a floating point unit would not be worth the expense in silicon area and design effort.

8. VCE design: The VCEs are designed at this step based on the detailed specifications. Internal register and data path widths are set according to the fixed point analysis. The state machines or micro code is defined for each task. The design was done using the Verilog hardware design language.

9. Generalization of VCEs so as to maximize flexibility with minimum added hardware cost and design effort. After the hardware designers defined the data paths required to perform the given tasks, the designs were shown to the algorithm developers who then proposed extra uses of the same data paths which would require only simple modification. For example the smoothing filter used to create the Gaussian pyramid was made more general to include any arbitrary separable 5×5 filter.

10. Finalize the specification of the VCEs: This step of 'design freeze' is a major milestone in the design.

11. Final design of the VCEs and other hardware blocks such as the DMA controller and the video interface.

12. Testing the hardware blocks on FPGAs: Once the VCEs were designed they were synthesized on an FPGA and their operation was tested together with the ARM CPU. Only one or two VCEs could fit onto a single FPGA so multiple FPGAs were required. This means that one could not create an exact FPGA model of the chip.

13. Integration and testing of logic design. This is the step of putting all the pieces together: the CPU cores with their memory caches, the VCEs, the DMA controller, memory buses and IO modules. At this point there is a complete Verilog model of the chip. This model can even run programs exactly as they would on the chip. However, it is very slow and a single frame takes a few hours to run. Test vectors (derived from actual image sequence data) were used to test the individual VCEs together with the CPU and the DMA channels. The results were compared to the emulator.

Following the logic level design (the 'front end' process described above) is the 'back end' process where the chip is laid out. At this point chip timing can be verified. Timing problems can lead to some redesign of specific logic paths. Details of this process is beyond the scope of this paper. When all timing problems are resolved then comes the manufacture and assembly.

3.1 The Hardware architecture design

This section describes in more depth the algorithm task requirements and how they led to the choice of basic chip architecture. The typical automotive vision application is made up of a set of tasks which must run every frame. Profiling of the algorithms running on a standard serial computer highlighted the computationally expensive tasks. These tasks were candidates for hardware implementation. Further analysis shows that each task has different characteristics requiring different types of computation:

- **Video input:** from one or multiple channels including gamma correction and color space conversion. This task is sequential and works at pixel clock rates (on the order of 10MHz).

- **Basic image processing:** functions such as constructing an image pyramid for multi-resolution analysis, image derivatives and edge detection. This task typically requires running 5x5 separable filters on the image and then applying thresholds, sub-sampling and possibly local maximum detection.

- **Road and lane analysis:** Due to the strong perspective in the image this task requires a row-by-row analysis with different parameters for each row. It does not map

well to SIMD architecture and more suitable to a serial DSP.

- **Pattern classification:** Following an offline learning stage, pattern classification algorithms such as template matching, Support Vector Machines and Neural Networks typically boil down to many vector dot products. This can be well addressed with SIMD MAC (multiply accumulate architectures) machines. For efficient implementation a large local memory is required for the templates, support vectors or neural network weights.

- **Tracking objects detected in previous frames:** The tracking algorithm is based on the SSD tracking of Lucas and Kanade[1]. There are three main parts to the algorithm: (1) an image warp and computation of the time difference image ΔI, (2) summing up the spatial and time derivatives, (3) inverting an $N \times N$ matrix to compute the motion estimate. Steps (1) and (2) map well to a SIMD architecture but require large fixed point precision. Step (3) is more suitable for implementation on a general purpose CPU.

- **Tasks with complex program flow:** Some tasks that are not suitable for hardware implementation are best implemented on a general purpose CPU. For example the results of vehicle tracking are used to compute target range and range rate [4] and forward collision warning [2]. To accommodate these tasks a second CPU was included.

Each task requires its own unique type of computation and since each task must be run every frame a parallel architecture was used as shown in figure 5. By having all the tasks running in parallel rather than sequentially the overall clock rate can be kept relatively low which is more acceptable in the more conservative automotive market.

Each VCE has its own local memory area. These memory areas range from 8KB to 64KB. Data transfers to and from the local memory are performed by the DMA engine and are initiated as needed by the VCE itself without CPU intervention.

4 Summary

The paper describes the EyeQ chip and the design process. The EyeQ chip was delivered to Mobileye in April 2004. It currently supports lane, vehicle and pedestrian detection applications. It offered enough flexibility to accommodate the algorithm progress that occurred since the design freeze step. New applications are being targeted for the chip which can make use of its computational ability.

References

[1] B.D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision", in *Proc. of the 7th International Joint Conference on Artificial Intelligence* 1981, Vancouver, pp. 674–679.

[2] O. Mano, G. P. Stein, E. Dagan and A. Shashua, "Forward Collision Warning with a Single Camera", in *Proc. of the IEEE Intelligent Vehicles Symposium (IV2004)*, June 2004, Parma, Italy.

[3] A. Shashua, Y. Gdalyahu and G. Hayon, "Pedestrian Detection for Driving Assistance Systems: Single-frame Classification and System Level Performance",in *Proc. of the IEEE Intelligent Vehicles Symposium (IV2004)*, June 2004, Parma, Italy.

[4] G. P. Stein, O. Mano and A. Shashua, "Vision-based ACC with a Single Camera: Bounds on Range and Range Rate Accuracy", in *Proc. of the IEEE Intelligent Vehicles Symposium (IV2003)*,June 2003, Columbus, OH.