

*A reconfigurable embedded vision system
for advanced driver assistance*

**Gorka Velez, Ainhoa Cortés, Marcos
Nieto, Igone Vélez & Oihana Otaegui**

**Journal of Real-Time Image
Processing**

ISSN 1861-8200

J Real-Time Image Proc
DOI 10.1007/s11554-014-0412-3



Journal of

**Real-Time
Image Processing**

JRTIP

 Springer

 Springer

Your article is protected by copyright and all rights are held exclusively by Springer-Verlag Berlin Heidelberg. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com".

A reconfigurable embedded vision system for advanced driver assistance

Gorka Velez · Ainhoa Cortés · Marcos Nieto ·
Igone Vélez · Oihana Otaegui

Received: 12 September 2013 / Accepted: 6 March 2014
© Springer-Verlag Berlin Heidelberg 2014

Abstract Computer vision technologies can contribute in many ways to the development of smart cities. In the case of vision applications for advanced driver assistance systems (ADAS), they can help to increase road traffic safety, which is a major concern nowadays. The design of an embedded vision system for driver assistance is not straightforward; several requirements must be addressed such as computational performance, cost, size, power consumption or time-to-market. This paper presents a novel reconfigurable embedded vision system that meets the requirements of ADAS applications. The developed PCB board contains a System on Chip composed of a programmable logic that supports parallel processing necessary for a fast pixel-level analysis, and a microprocessor suited for serial decision making. A lane departure warning system was implemented in the case study, obtaining a better computational performance than the rest of the works found in the literature. Moreover, thanks to the reconfiguration capability of the proposed system a more flexible and extensible solution is obtained.

Keywords ADAS · System on Chip · Hardware/software codesign · Embedded systems · Smart cities

1 Introduction

Together with the massive proliferation of smart devices across different scales, the idea of a smart city has moved from science fiction towards reality. The concept of smart city emerged the last decade as a mix of ideas about how information and communication technologies (ICTs) might improve cities, enhancing their competitiveness and efficiency, and providing new ways to solve poverty, social deprivation and poor environment problems [12]. The term smart city has many definitions, but in all of them the ICTs are central to the operation of the future city [4]. The ICTs are changing cities by making them smart, not only in the terms of automating routine functions serving individual persons, buildings or traffic systems, but in ways that enable us to monitor, understand, analyse and plan the city to improve the efficiency, equity and quality of life for its citizens in real time [5].

Among the different ICTs, the computer vision technology can make a major contribution to the development of smart cities. Applications such as traffic surveillance [6, 23], vehicle classification [8] or advanced driver assistance systems (ADAS) [10, 11, 27] are becoming more and more popular.

In particular, ADAS applications are expected to experience a significant growth in the following years, from a demand of 60 million ADAS units in 2013 to a demand of more than 100 million units in 2018. Furthermore, the market research firm ABI Research predicts that the 60 % of the world's cars and the 80 % of the North American and Western European cars will include features like built-

G. Velez (✉) · M. Nieto · O. Otaegui
Vicomtech-IK4, Paseo Mikeletegi 57, Donostia-San Sebastián,
Spain
e-mail: gvelez@vicomtech.org

M. Nieto
e-mail: mnieto@vicomtech.org

O. Otaegui
e-mail: ootaegui@vicomtech.org

A. Cortés · I. Vélez
CEIT and Tecnun (University of Navarra), Manuel de Lardizábal
15, Donostia-San Sebastián, Spain
e-mail: acortes@ceit.es

I. Vélez
e-mail: ivelez@ceit.es

in Internet and smartphone connectivity in 5 years [26], which increases even more the existing application possibilities.

There are many design options when it comes to develop an embedded ADAS device, but in any case, some minimal requirements have to be fulfilled apart from the obvious real-time performance. Nowadays, embedded systems and software correspond some 30–40 % to the total value of the vehicle [7], and this number is expected to grow. Thus, it is necessary to design a low cost hardware compatible to the maximum number of ADAS vision applications. It is also necessary to obtain a small sized low power solution. This is especially important for electric vehicles, where the total weight of the on-board computing and communication systems and their power consumption are critical factors [28]. Furthermore, the design cycle for electronics and auto makers has been shortened from the 60-month cycle of about 5 years ago, to the 24–36 months of nowadays [26]. So a platform that can expedite the development process must be found. Not only this, due to their high cost, people cannot change vehicle frequently, so if we want to vehicles be part of the smart city and given the rate at which ICTs develop, the designed device should be able to be updated easily.

There have been different approaches in the literature to implement vision algorithms in embedded systems. For many years, only microcontrollers and microprocessors were used, due to their programmable functionalities [18]. There are many examples in the literature of vision systems with a microcontroller or microprocessor implementation, such as [1] or [33]. A similar approach is the use of digital signal processors (DSPs) [17, 34, 37]. DSPs can do single cycle multiply and accumulation operations and have some parallel processing capabilities, which enhance processing speed. Traditionally, DSPs have been used in image and audio signal processing when the use of microcontrollers was not enough.

However, the increase in resolution and frame rate in recent cameras makes it difficult to achieve real-time performance using only microprocessors or DSPs. Hardware implementation represents an alternative solution, since it can achieve a much better computational performance. During last years, there have appeared a big number of implementations in Application-Specific Integrated Circuits (ASIC) [9, 19] or field-programmable gate arrays (FPGAs) [2, 13, 16, 22, 29, 36]. FPGAs have an important advantage over ASICs: they are reconfigurable, which gives them some of the flexibility of software.

As time goes on, more complex ADAS applications are demanded. They not only need to do image processing, but they also need to communicate with other devices and offer a usable user interface. Developing the whole application in hardware is a too cumbersome task, so hybrid solutions

have appeared which combine software and hardware implementation.

The work in [31] describes a System on a Chip (SoC) designed to support a family of vision algorithms. However, the system uses an ASIC, so it cannot be reconfigured completely. This important drawback limits the capacity of the device from being updated. In [14], a lane departure warning system (LDWS) is implemented on an image-processing platform. The developed custom board can only be used for testing purposes, and the obtained performance is not enough for a more advanced driver assistance application. In [15], another LDWS is presented based on software and hardware codesign. However, the FPGA is only used to capture images from the camera, missing its computational potential for image processing. More recently, a generic embedded hardware and software architecture was proposed to design and evaluate ADAS vision applications [3]. Although the system is useful to test some vision applications, the performance obtained in the case study shows that the system is not enough to run more complex applications that include I/Os management, vehicle communications and other more demanding vision applications.

This paper presents a novel miniaturised embedded vision system suited for ADAS applications that is more computationally efficient than the rest found in the literature. The proposed system uses a Xilinx Zynq-7000 SoC, which is formed by an FPGA that harbours a dual core ARM. The use of such a system increases significantly the communication bandwidth between programmable logic's parallel computing and processor's serial computing, which is a great advantage over traditional hardware and software configurations, such as two-chip solutions or FPGAs with soft cores. Furthermore, the system is prepared to be easily reconfigured, which is another major advantage. The system was evaluated in a case study where an LDWS application was implemented in the developed PCB board using a hardware/software codesign methodology. The obtained computational time allows the system to include another more complex vision algorithm running in parallel.

The rest of the paper is structured as follows. Section 2 explains the methodology used to design the proposed system. Section 3 describes the system, and Sect. 4 presents a case study to show how the proposed system is suitable for an ADAS vision application. Section 5 presents a discussion, and finally, Sect. 6 concludes the paper.

2 Design methodology

In traditional design methodologies for embedded systems, hardware decisions are taken first. After an initial

specification, the hardware architecture is designed based mainly on the experience of the hardware designer team. This methodology has some important drawbacks. First, it can delay software teams, because in some cases the software cannot be started to develop and test until the hardware design is available. Furthermore, it can also delay the whole product design chain if a critical hardware design error is detected late. Finally, there is a risk for overdesigning or underdesigning the system due to a lack of an initial evaluation of software's computational requisites.

To improve the design chain, a different design methodology is proposed here, based on modern hardware/software codesign methodologies [3, 21, 32]. Before designing the final platform, a functional software prototype is developed using a more flexible platform: a standard PC. After analysing and evaluating this software prototype, the final platform's architecture is designed.

More specifically, the proposed design methodology consists of the following steps:

1. Determine the aims and the behavioural specification of the whole system. In our case, the aim is to develop an embedded device that could help to obtain a better quality of city life improving the road traffic safety. This device should be flexible enough to admit different ADAS applications. In this step, the behaviour of the product is defined as a specification that will guide our work in the future. Several techniques can be applied here, such as brainstorming, questionnaires among potential users or focus groups. It is important to reflect in the product the needs of the user, aligned with the main aims defined initially.
2. Develop a first functional software prototype in a PC. Although some classic methodologies left prototyping for the final steps, other more modern approaches, such as design thinking, promote prototyping from the beginning stages of the design process [25]. This prototype can serve as a functional, but already executable, specification of the system. In case of a vision system for ADAS, the prototype can be simplified using generic hardware, such as a PC. The main input, the camera, can be replaced by previously recorded videos, and the whole application can be programmed using a single programming language, such as C++. In this way, the programmed algorithms can be debugged more easily, in a more controlled environment.
3. Define the embedded system's architecture. In this step of the design, hardware and software teams work together defining the architecture of the final platform. This architecture must satisfy the design constraints for real-time embedded systems: cost, size, real-time performance and power. Additionally, the platform must provide a fast development cycle. This last consideration is crucial, as time-to-market is a key factor in the automotive industry. Analysing the algorithm features and the computational requirements of the previously developed software prototype, an exploration of potential solutions can be made. In the case of an embedded real-time vision system for driver assistance in smart cities, we propose the use of an SoC architecture. Taking into account that real-time image processing is a time-consuming task, a hybrid SoC architecture can speed up significantly the execution time. In an SoC, you can combine a programmable logic and a processing system. Furthermore, the reconfigurable feature of SoC architectures can be exploited to obtain a smarter solution. This stage includes an analysis of the algorithms to decide which parts require hardware implementation and which parts are run in a microprocessor. The software prototype is written in C++, so it is easy to port it to the microprocessor and to measure the computational times of its different parts. Normally, the most time-consuming parts are composed of pixel-level operations that can be significantly accelerated with a parallel implementation in an FPGA. On the contrary, other parts are better described in C++ and it is not worthwhile porting them to an FPGA. The decisions taken in this design step are crucial. Here, system's IOs and the communication channels of the different modules are defined. As we are using reconfigure hardware, some design errors are easy to solve in the future, but in any case, it is fundamental to spend the necessary time here to make a good architecture specification.
4. Develop the final application. Once the architecture is defined, the hardware team designs the board of the embedded device. The hardware team is also responsible for porting the previously selected software modules to the FPGA. Meanwhile, the software team modifies the source code to adapt it to the new architecture.
5. Validate the embedded device. The software prototype developed previously is used as a golden reference model to compare to the results of the application under test. This reference model together with simulation tools can help reveal errors very early in the test cycle. Once all the detected errors are fixed using videos as input, final validation tests are run in the real environment. In the case of an ADAS device, the device is installed in vehicles and it is tested in a variety of driving conditions.

This design flow should not be considered a rigid waterfall design. The last two steps on the design chain are run

iteratively until a satisfactory solution is reached. Additionally, an error found in the validation step can lead to a redesign, and decisions taken during Hardware/software partition may also change the design.

3 System overview

Two external interfaces can be highlighted in a vision system for driver assistance. First, a camera is needed to acquire the images. Secondly, these systems usually receive vehicle information which is helpful to enhance the system behaviour. For this purpose, normally Controller Area Network (CAN) communications are implemented. These two interfaces are enough for vision systems which focus only on the image processing. Nevertheless, the system maintenance and upgrade are also very important issues to achieve a competitive and flexible product. Ethernet communication is a good candidate to carry out these operations since it can make the management easier.

Taking into account aspects such as costs, power consumption, computational performance, design flexibility and time-to-market, the Xilinx Zynq-7000 Extensible Processing Platform (EPP) has been selected as our system platform. The Zynq-7000 provides internally several peripherals which facilitate the interface communications,

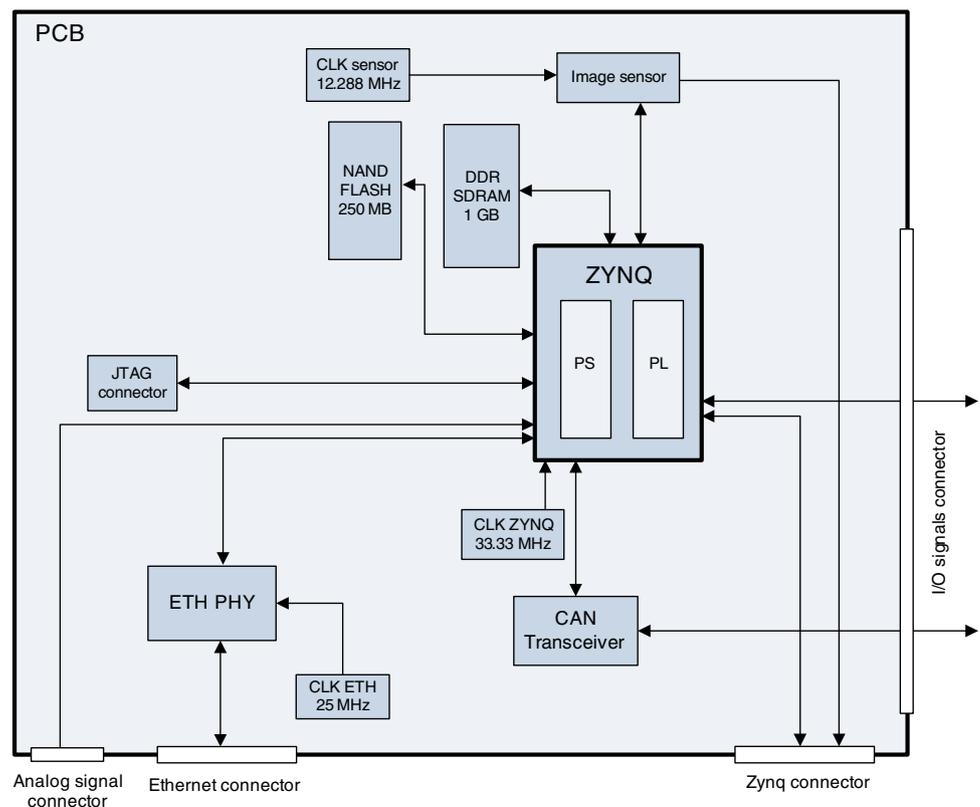
such as Ethernet or CAN, reducing the number of external components in the overall system. Furthermore, the Zynq-7000 EPP is an SoC which achieves the desired software and hardware codesign.

3.1 Hardware

The proposed system design is oriented towards an efficient platform for different low-cost embedded ADAS such as an LDWS, a Traffic Signs Recognition System or a Driver Fatigue Detection System. According to the required FPGA resources, different devices can be found within the Zynq-7000 SoC family. To get a trade-off between system costs and FPGA resources, the XC7Z020 has been selected. This device is based on Artix-7 technology, which is the cheapest one among the Series 7 FPGAs, but has all the necessary FPGA resources for this kind of systems.

Figure 1 presents the proposed system architecture. The Zynq-7000 EPP is composed of a Processing System (PS) and a Programmable Logic (PL). The Dual-core ARM Cortex-A9 and their peripherals are in the PS part whereas the Field Programmable Gate Array (FPGA) is in the PL part. Both the PS and PL have I/O ports to achieve external communications. An I/O signals connector is employed to be able to send and receive external digital signals. Some environment information such as rain or light can also be

Fig. 1 System architecture of the Zynq-7000 EPP



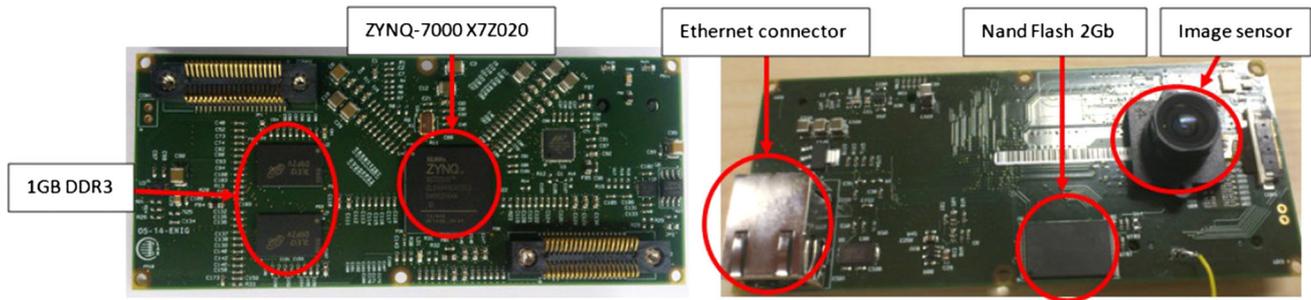


Fig. 2 Developed PCB board: PCB Zynq (*top and down*)

received through the Analog signal connector. Furthermore, an additional Zynq-7000 EPP can be connected through the Zynq connector to increase the system capability. Thus, the system can cover advanced processing integrating several ADAS in the same platform with a master-slave configuration. Both platforms will be able to communicate through an SPI interface and the Master Zynq-7000 EPP will be responsible for managing all the system outputs.

The developed PCB uses two external memories: a 250 MB NAND Flash and a 1GB Double Data Rate Synchronous Dynamic Random-Access Memory (DDR SDRAM) as can be observed in Fig. 1 and also in the photos presented in Fig. 2. The NAND Flash has been selected as the Non-Volatile Memory because, even though it has a high number of pins, the NAND Flash provides the density needed for this kind of applications. On the other hand, the DDR SDRAM can be used to store the ARM application and the embedded Operating System (OS). PS and PL share the DDR controller, so that both of them can access the DDR SDRAM and share data between them. The total size of the developed PCB is 120×60 mm.

3.2 Software

The proposed system not only has to execute image processing algorithms but also has to use communication and network interfaces, and it has to be able to be updated externally. In light of this, an Embedded Linux operating system has been chosen to run in the ARM. The performance decreases when using an application that runs on a operating system instead of using a standalone application, but on the other hand, there are great savings in development time and in the maintenance of the system. There is a big number of communication drivers for Linux, which make the development of CAN or Ethernet applications much easier. It is worth losing some computational performance to shorten substantially the development time and to gain flexibility. Furthermore, when using an operating system, the programmers can focus on the specific

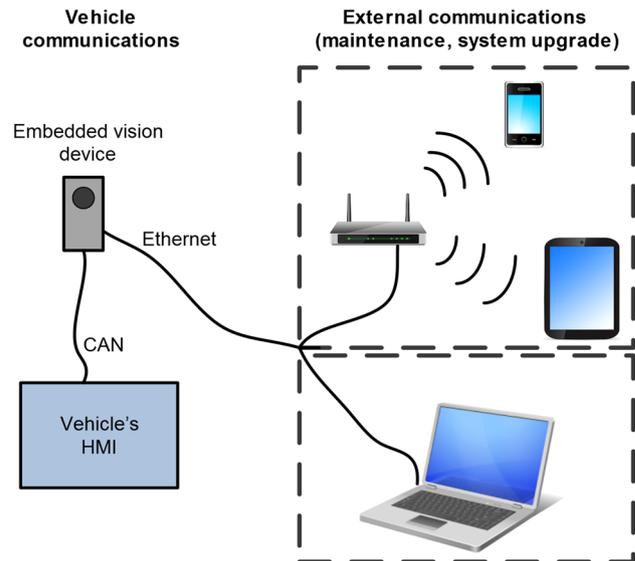


Fig. 3 Device communications

image processing algorithms without having to care about other low level details. A higher abstraction level reduces programming errors and makes the source code more portable.

3.3 Reconfiguration capability

One of the main strengths of the proposed embedded system is that it is easily reconfigurable. The Zynq's PL is programmed via PS using a bitstream file. On each startup, the PS programs the PL using the bitstream stored in the NAND Flash. Thus, the way to reconfigure the PL changes that file. A Linux operating system is running in the PS, so it is easy to create an application that is listening for a reconfiguration request to update the bitstream file. As explained before, there are two main processes running in the installed operating system. One of them is in charge of the image processing, and the other one is in charge of the decision making and the I/Os. The second process will be

constantly listening for any connection request through Ethernet. Once properly authenticated, we can upload a new bitstream file from a laptop, or from any other smart device such as a mobile phone or a tablet. In the last cases, a Wi-Fi access point shall be created using an intermediate router. Figure 3 depicts the embedded device's communications with other devices. The duration of PL reconfiguration is in the order of seconds, so it is not a critical delay, specially if the reconfiguration is done when the vehicle starts.

As well as reconfiguring the PL logic, it is also possible to update the installed application in the ARM. The initial RAM image with the application binary is also stored in the NAND Flash together with the Linux image. On each startup, the bootloader reads these images and loads them into DDR SDRAM. In the same fashion as the PL's reconfiguration, we can upload a new version of the Linux kernel or a new version of the ADAS application that runs on the PS. We only need to modify the files stored in the NAND Flash, and after resetting the device we will have a newly updated device.

There are potential security threats that should be considered in the development of an embedded device, such as the cloning of the embedded device, the access to privacy data inside the embedded system or the insertion of malware. Xilinx offers methods to obtain a secure boot with Zynq-7000 using public and private cryptographic algorithms [24]. Furthermore, the PS-PL architecture provides the programmer with the ability to provide redundancy in recovering from operational failure in either the PS or PL. The PS can monitor the PL and vice versa.

4 Case study: lane departure warning system

The hardware and software architectures described in the previous section are suitable for a variety of ADAS vision applications. This section presents a case study for the implementation of an LDWS.

The design of this specific application was done following the design methodology explained in Sect. 2. First, the aim and the behaviour of the system were specified. In this case, the aim is to warn the driver for unintentional lane departures to avoid possible accidents. Thus, the solution should be able to detect and track in real-time the road's lane markings and determine the position of the vehicle inside its own lane.

Once the complete behaviour specification was done, a functional software prototype was developed in a PC. This prototype was tested using videos recorded on real scenarios. During the tests, the algorithm was adjusted until the behavioural specification was fulfilled.

On the next step of the design chain, how this prototype would be implemented in the embedded device was decided. The whole algorithm was first implemented in the ARM to analyse which modules required hardware implementation and which modules could be left in the ARM. After this analysis, the final application was implemented and validated in the developed PCB board.

Following this introduction comes a brief description of the Lane Departure Warning algorithm. Then, the hardware-software partitioning of the algorithm is explained. Finally, the hardware and software used in the application are detailed and the obtained results are discussed.

4.1 Algorithm description

The LDWS is one of the most widely used ADAS application. There are many implementations of this ADAS application in the literature. In our case, we have implemented a vision system that models in real-time the lane markings using a single camera [20].

The method receives as input the sequence of frames captured with the camera and generates as output a road model at each point in time. Figure 4 shows a block diagram of the algorithm. First, there is a pre-processing stage where the saturation of the image is analysed to feedback dynamically the camera parameters. Then, the image is filtered using a step row filter to detect lane markings.

The step row filter is an 1D convolution filter that operates at each row of the original image. Its aim is to determine which pixels of the image likely represent lane markings. The filter has been designed to be adapted to the perspective of the scene so that the convolution mask is different for each row of the image. The filter function is defined as: $y(r, c) = 2 * x(r, c) - x(r, c - \tau(r)) - x(r, c + \tau(r)) - \text{abs}(x(r, c - \tau(r)) - x(r, c + \tau(r)))$, where r stands for row and c for column. $x(r, c)$ is the pixel value at (r, c) and $\tau(r)$ is the tau value for the given row. This variable is defined according to the perspective so that it is larger at the bottom of the image and smaller as it get closer to the vanishing point y value (more details can be found at [20]). The filter generates high values for pixels whose intensity value is higher than its lateral neighbors (at $\pm \tau(r)$) and these side values are similar. This provides high values for stripes like lane markings and generates very few false positives for other bright visual patterns that might appear in vehicles or in the road.

The lane markings pixels are then accumulated using a perspective LUT histogram. The search for maxima leads to the generation of linear measurements of the state vector that are fed into the Kalman filter tracking. Finally, there is a semantic analysis that has as a result the information to activate or deactivate the lane departure warning signal.

Fig. 4 Diagram of the implemented LDWS algorithm

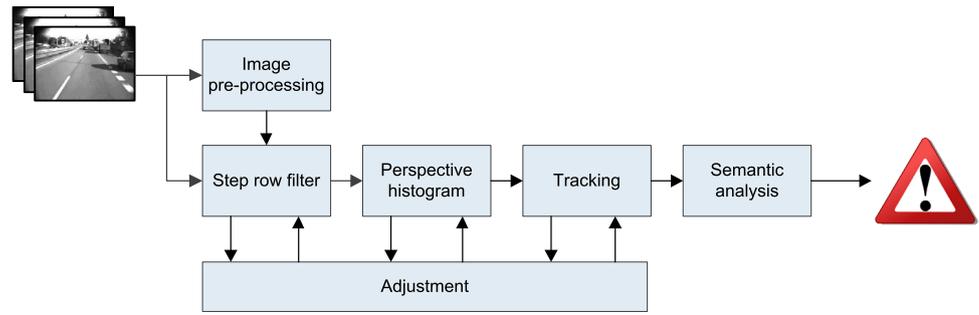
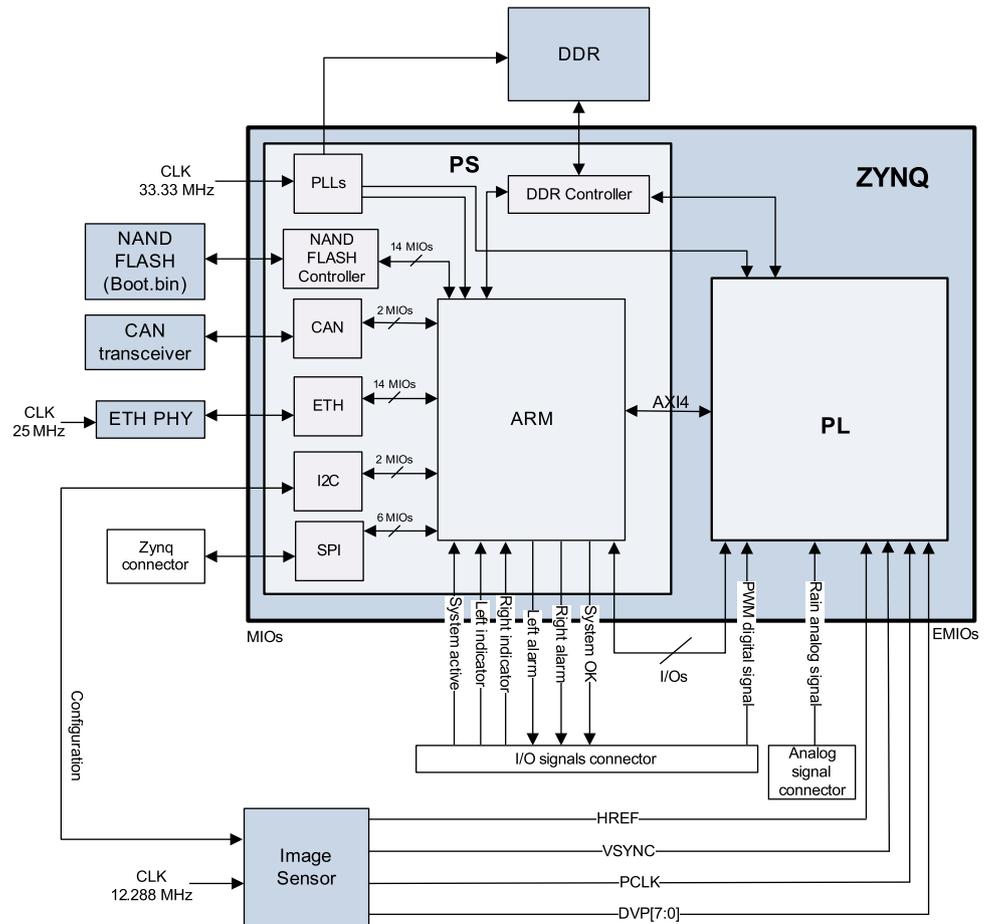


Fig. 5 Interfaces for the low-cost embedded LDWS



4.2 Hardware/software partition

In this kind of applications, a frame rate of 25 fps can be considered as a real-time processing. Therefore, the maximum time to process one frame is 40 ms to achieve a real-time system. The proposed system must fulfill some system requirements fixed by an Industrial Project taking into account other processes used for decision making and I/Os. These processes are out of the scope of this paper. However, they are executed by the ARM in the PS consuming a timing of 10 ms. Therefore, a 25 % of the timing must be reserved for this purpose. Additionally, our system should

be able to execute in parallel the LDWS and another more computational expensive application like a Collision Avoidance System. Then, 20 ms, which is a 50 % of the overall timing, must be reserved for this additional costly application.

To achieve a real-time system, the image processing of the LDWS is required to be executed as maximum in 10 ms. This timing budget helps us to decide the necessary SW/HW partitioning. On the one hand, the camera interface should be implemented in HW to accelerate the pixel-level image capture. Nevertheless, this interface has to include a data binning to obtain the appropriated

resolution. On the other hand, the step row filter, required in the LDWS, is also a pixel-level processing that consumes most of the total computational time when running the whole algorithm in the ARM. Its parallelisation is straightforward. Therefore, it should be implemented in HW, that is, in the PL. The rest of the algorithm can be implemented in SW, in the ARM of the PS.

4.3 Hardware

There are different modes to communicate PS and PL as can be seen in Fig. 5. They share the DDR controller so that both of them can access the DDR SDRAM. PS and PL can also communicate internally by the Advanced eXtensible Interface (AXI) standard. The AXI is part of the Advanced Microcontroller Bus Architecture (AMBA) and this standard searches a higher performance and an unified interface. The AXI4 specification is similar to AXI but with three sub-interfaces: Memory Map (traditional address/data/control interface), Stream (high performance data streaming) and Lite (simple interface reducing logic requirements).

The I/O ports of the PS are called Multiplexed IOs (MIOs), and the Extended Multiplexed IOs (EMIOs) correspond to the I/O ports of the PL. The MIOs can be connected to EMIOs to get dedicated communications between the PS and PL. This communication through external ports (MIOs and EMIOs) can be very useful, for example, for test purposes. On the other hand, some MIOs are used to connect the ARM to the required PS peripherals or as specific digital I/Os. In the LDWS, the following specific digital I/Os can be distinguished:

- System active: is the button input which indicates the system activation.
- Left indicator: is the left vehicle indicator which marks a voluntary lane change.
- Right indicator: is the right vehicle indicator which marks a voluntary lane change.
- Left alarm: is an output to warn an involuntary lane change towards the left.
- Right alarm: is an output to warn an involuntary lane change towards the right.
- System OK: is an output to alert the driver that the system is working correctly.

The Left alarm and Right alarm signals of the proposed system are well known as the lane departure warning signals of the LDWS.

The embedded LDWS employs several PS peripherals as shown in Fig. 5. For instance, the phase-locked loops (PLLs) are used to generate from a source clock of 33.33 MHz the necessary clock frequencies for the DDR SDRAM, the ARM processors and the PL. In LDWS, the

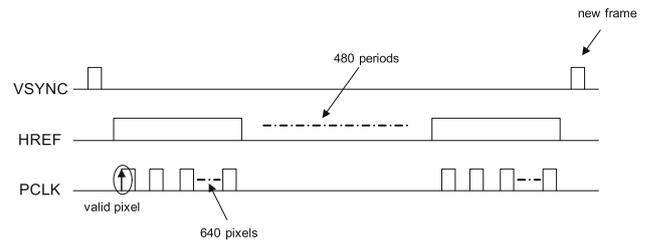


Fig. 6 Timing diagram of the CMOS image sensor

PL employs clock frequencies of 100 and 200 MHz, the ARM processors work at 666.66 MHz and the DDR SDRAM uses a clock of 533.33 MHz. However, these values are soft configured.

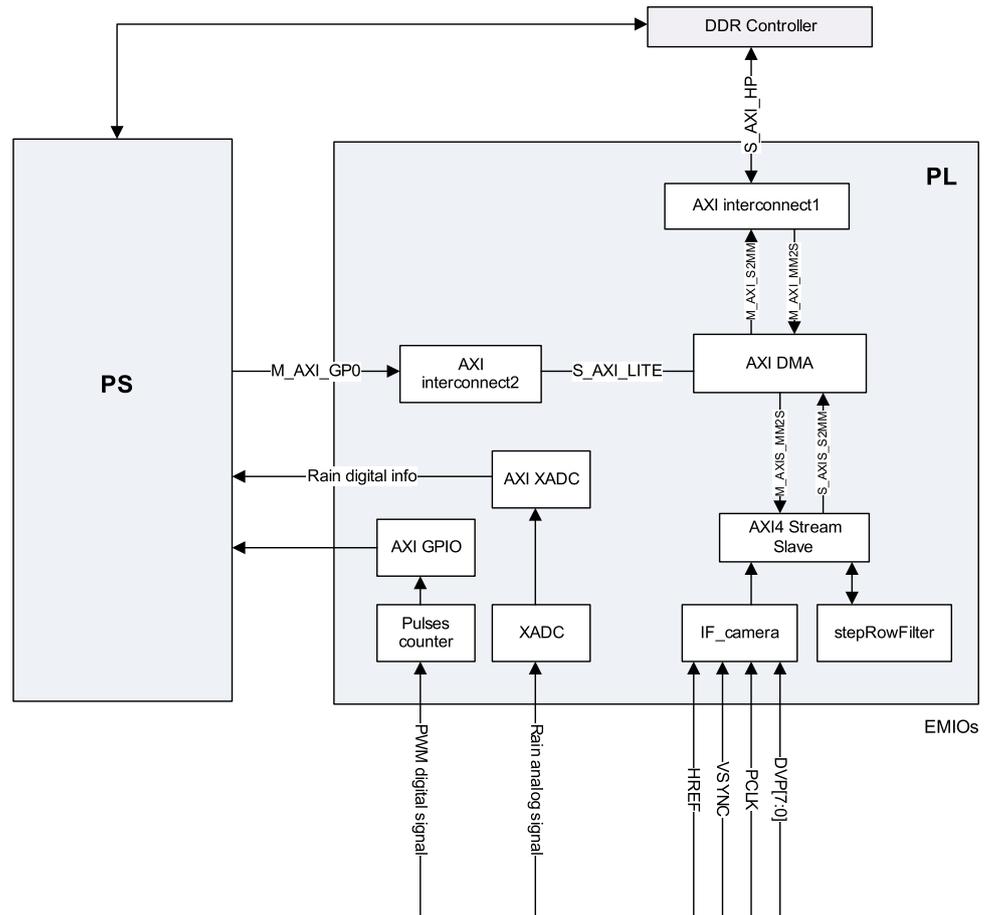
The configuration of the CMOS image sensor is carried out by the I2C Master peripheral of the PS. Nevertheless, the image sensor is also connected to the EMIOs in the PL since the image capture is done by the FPGA, which is faster than a software capture [15]. Once the image sensor is calibrated, it starts the normal operation shown in Fig. 6. The timing diagram of Fig. 6 presents the acquisition of a 640(H) × 480(V) image. A pulse in VSYNC signal indicates that a new frame starts. HREF is a logic one during 640 cycles of PCLK. A rising edge in PCLK signal marks that a pixel is valid. Thus, 640 horizontal pixels are obtained. To complete the image capture, 480 periods of HREF are needed. The LDWS requires a 320(H) × 240(V) image to process the algorithm. Therefore, the received image will be preprocessed before storing in the DDR.

4.3.1 PL implementation

VHDL is used to describe the designed logic in PL. There are several tools that can generate VHDL code from C++ code, such as Matlab or Vivado HLS. These tools can speed up the development process; however, they are not always the best choice [30, 32]. Some important sources of parallelism such as complex loop specifications cannot be efficiently parallelizable [32]. In our case, the VHDL code has been hand coded since the safety requirements of our system do not permit us to use this kind of automatic generation tools.

ISE Design Suite 14.5 tools of Xilinx have been used to simulate (ISE simulator Isim), synthesize, place and route (PlanAhead tool) our VHDL design. Additionally, EDK tools such as Xilinx Platform Studio (XPS) and Xilinx Software Development (XSDK) have been employed to define the hardware platform and develop the software in PS.

The architecture of the VHDL implementation in PL is shown in Fig. 7. The PS configures the registers of the Xilinx AXI DMA module through AXI LITE interface,

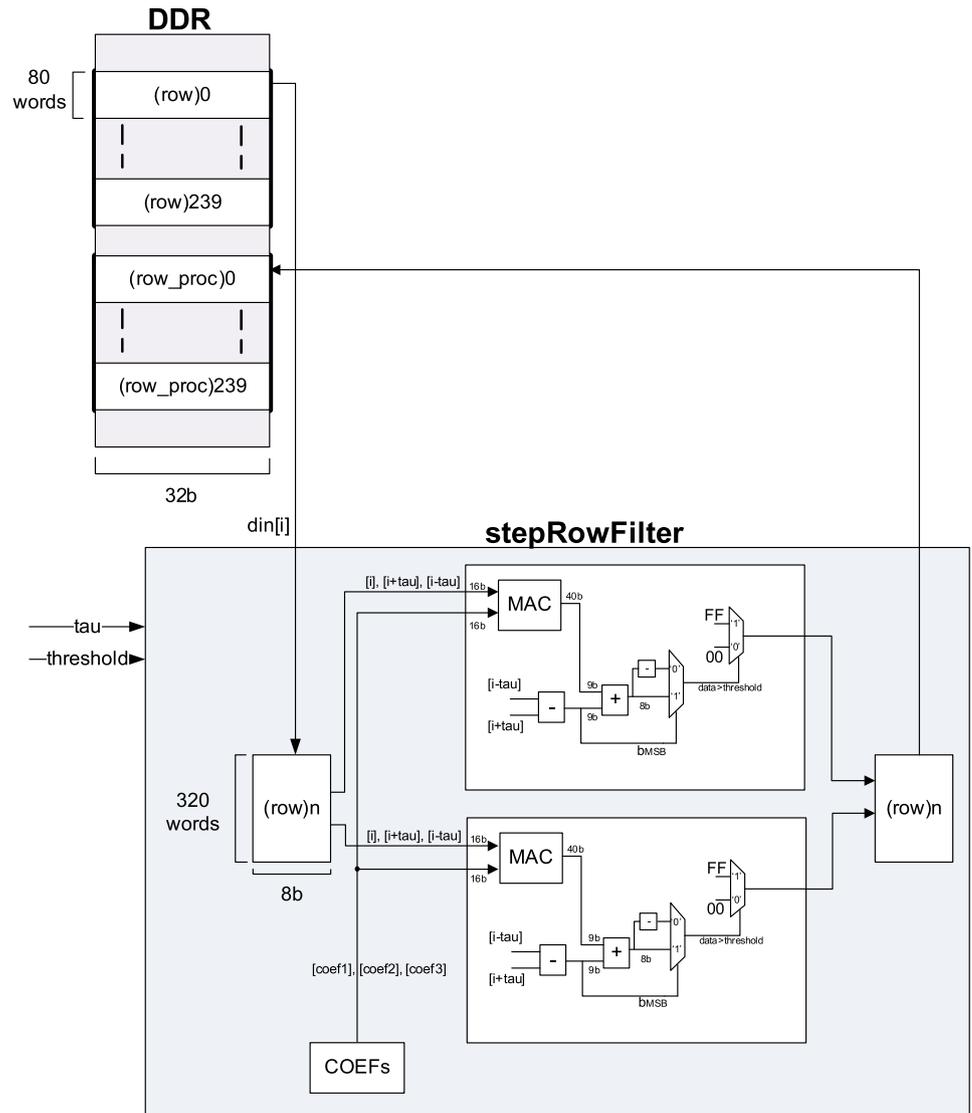
Fig. 7 Architecture of the PL for the LDW system

which uses less lines than the AXI interface and is enough for this configuration. Our IF_camera module receives the camera data and processes the data binning to achieve the desired $320(H) \times 240(V)$ image from the received $640(H) \times 480(V)$ image. When two rows of the image have been received, the obtained 320 pixels are stored in Block RAMs (BRAMs) of the PL. These BRAMs are read by the AXI4 Stream Slave which sends the information to the S_AXIS_S2MM channel of the Xilinx AXI DMA module. It manages the addressing of the DDR SDRAM in an efficient manner. The M_AXI_S2MM channel is the responsible for sending the pixel data to the DDR Controller using the S_AXI_HP channel which is the fastest path to access the DDR from the PL. Thus, the captured frame is stored in the DDR SDRAM. Then, the PS reads the stored image, carries out a saturation analysis applying some statistical operations over it and saves it into the DDR SDRAM again. After these operations, the PL reads the preprocessed image from the DDR SDRAM using the M_AXI_MM2S channel of the Xilinx AXI DMA module and our Step Row Filter module applies the custom filter to the image. Finally, the filtered image is stored again in the DDR SDRAM so that the PS reads the image completing

the LDW algorithm. Additionally, the PS is in charge of activating the system outputs according to the algorithm results.

Figure 8 shows an scheme of the stepRowFilter module architecture. Each row of the $320(H) \times 240(V)$ image is stored in PL. The module uses two MAC (Multiplier Accumulator) units to process two pixels in parallel. The selected pixels will be read from the BRAM depending on the value of τ which is sent by the PS and they will be multiplied by the coefficients stored in the LUT applying the corresponding mask. Each MAC unit accumulates the multiplication of three pixels by the three coefficients. The result of the MAC unit is added by the subtraction of the pixels $[i - \tau]$ and $[i + \tau]$. The sign of this sum depends on the sign of the subtraction result. Finally, data are compared with a threshold fixed also by PS. Thus, the output is limited: if data are higher than the threshold, the output will be "FF" and if not, it will be zero.

In addition to the camera data, PL receives a PWM signal from a speedometer. To process this information, a design which counts the timing of this signal at low level has been implemented in the PL. Thus, the PL sends the corresponding code to the PS so that it is able to calculate

Fig. 8 Architecture of the stepRowFilter module

the vehicle speed. For this purpose, an AXI GPIO module is used to send this signal through the AXI interface. The PL also receives a rain analog signal from a rain sensor to get environment information which can enhance the system working. This signal is digitalised using one of the Xilinx Analog Digital Converters (XADCs) of the PL. The 12-bits digital data are sent to the PS using the AXI XADC module which provides the AXI interface to the XADC converter. Thus, the system costs are reduced considerably avoiding the inclusion of an external ADC.

4.4 Software

The software runs in a Xilinx Zynq Linux operating system within the ARM of the PS. This distribution's kernel is based on the 3.0 Linux kernel together with Xilinx additions. This distribution was chosen because it is lightweight and it has been designed specifically for the Zynq platform.

In the initialisation, the application running in the PS configures the camera. Once the camera is configured, the LDW algorithm starts running in one process. This process uses Viulib 13.10 libraries for some of its computer vision functions. Viulib is a multiplatform SDK written in C++ that simplifies the building of complex artificial vision applications [35].

In a separate process, the I/Os are managed and the main decisions are taken. Both processes communicate with each other using shared memory. When the LDW process has finished processing a frame, it writes the output in a certain position of the shared memory. Then, the decision making process reads this output and together with the information received from the vehicle it decides to activate or deactivate the lane departure warning signal.

As explained in Sect. 3.3, the software can be updated via Ethernet. Furthermore, the PS can reconfigure the PL to adopt new applications.

Table 1 Lane detection rates for different subsets of the considered sequences

		Length (min.)	Availability (%)	TP	FP	FN	R	P
Position	Bus	197'14"	87.48	355	44	22	0.94	0.89
	Car	139'26"	94.88	255	19	11	0.96	0.93
Light	Sunny	155'39"	91.72	288	28	11	0.96	0.91
	Cloudy	133'12"	87.26	259	23	18	0.94	0.92
	Dusk/dawn	46'22"	95.79	56	12	4	0.93	0.82
	Night	1'26"	99.00	7	0	0	1.00	1.00
Weather	Dry	236'00"	91.68	401	44	21	0.95	0.90
	Rain	100'40"	87.89	209	19	12	0.95	0.92
Road	Highway	220'30"	93.22	430	42	19	0.96	0.91
	Periurban	115'26"	85.51	179	20	14	0.93	0.90
Total		336'41"	90.55	610	63	33	0.95	0.91

TP true positives, FP false positives, FN false negatives, R recall = TP/(TP + FN), P precision = TP/(TP + FP)

4.5 Results

An LDWS has been successfully implemented in the proposed platform. From the first software prototype, the system was tested using as input videos recorded on real situations. The appearance of lane markings greatly varies depending on illumination, road condition and weather, so different videos were recorded taking into account the combination of these conditions. The set of sequences has a total duration of 336 min and was recorded with two types of vehicles: car and bus.

One way to objectively evaluate the system is to check the correct detection of lane changes. We hypothesize that the system is able to monitorise when the vehicle is getting out of its own lane if lane changes are detected correctly. We have also determined the availability of the system as a result of a self-assessment module which determines the reliability of the detections. Below a certain threshold, the system switches OFF, and the availability is measured as the percentage of time in which the system is ON.

Table 1 shows the statistics of lane change detections categorised according to position, light, weather and roadway type. The recall and precision values are computed to show the performance of the proposed method. The recall, which is related to the number of correctly detected events, is above 95 %, while the precision, related to the quality of the detections, is above 91 %. So it can be said that the algorithm overcomes the difficulties of identifying unintentional lane departures. Additionally, there is an ongoing evaluation which is being done in the real environment. In this case, the developed device is installed in a vehicle and the images obtained from the integrated camera are the inputs that feed the system. The preliminary results of this ongoing validation are positive and confirm the evaluation done previously using videos.

4.5.1 PL resources

Table 2 summarizes the PL resources used by our modules: IF_camera and stepRowFilter. In the IF_camera module,

Table 2 PL resources of the modules implemented in XC7Z020

	IF_camera module (%)	StepRowFilter module (%)
Slices	79 (1)	68 (1)
Slice registers	121 (1)	145 (1)
Slice LUTs	196 (1)	120 (1)
RAMB18E1s	–	2 (1)
RAMB36FIFOE1s	32 (23)	–
External IOBs	53 (26)	64 (32)
DSP48E1s	–	2 (1)

the received pixels to be processed are kept using Slice Registers. The Slice LUTs are used for implementing logic and also memory as shift registers to store the results of the binning operations. The IF_camera module needs 32 RAMB36FIFOE1s to store the 320(H) × 240(V) valid pixels which are obtained from the 640(H) × 480(V) pixels in the received image. The stepRowFilter module employs two DSP48s to process the filtering of two pixels in parallel. This custom filter is processed per row. So, in the PL only 320 pixels for the input and 320 pixels for the output are needed to be stored in BRAM, requiring two RAM18E1s.

4.5.2 Performance

In Table 3, a timing profile of the LDW image processing algorithm is presented when it is totally implemented in the PS. The necessary time to process the LDW algorithm over a 320(H) × 240(V) image is 18.902 ms. Taking into account the time budget defined previously, we only have the 25 % of the cycle to run the image processing of LDW, that is, 10 ms. Consequently, it is necessary to optimise the implementation using the PL.

Filtering two pixels simultaneously in the PL, the processing time of the Step Row Filter with a clock frequency

Table 3 Timing profile of the LDW image processing applied to a 320(H) × 240(V) image implemented in the PS

Module	Time (ms)	Percentage (%)
Image pre-processing	5.991	31.690
stepRowFilter	10.530	55.700
Perspective histogram	2.217	11.720
Tracking	0.163	0.860
Parameters adjustments	0.001	0.005
Semantic analysis	0.000	0.000
Total	18.902	

Table 4 Timing profile of the co-designed solution applied to a 320(H) × 240(V) image

Module	Time (ms)	Percentage (%)
Image pre-processing	5.991	62.427
stepRowFilter	0.950	9.899
PL-DDR communication	0.274	2.862
Perspective histogram	2.217	23.101
Tracking	0.163	1.698
Parameters adjustments	0.001	0.010
Semantic analysis	0.000	0.000
Total	9.596	

of 200 MHz is 0.95 ms. The overhead of the communication between PL and PS is 0.274 ms, which is the access time needed to communicate PL with the DDR in different stages. As it can be seen in Table 4, the overall processing time of the image processing is below the target 10 ms, achieving a time reduction of around 50 %. Furthermore, the filtering parallelisation can be increased considerably depending on the resources of the target FPGA. This parallelisation permits to process more pixels simultaneously. In that case, the processing time reduction would be even higher.

The obtained computational time meets the defined time budget constraint for the LDW application. The image processing part requires 9.596 ms of the period, and another 10 ms are reserved for the decision making, I/Os and other necessary functions. Additionally, 20 ms are reserved to implement another more computational expensive application like a Collision Avoidance System.

4.6 Comparison

Table 5 shows a comparison of the computational performance of our LDWS implementation with other works of the literature that implement an LDWS on an embedded vision system based on a hybrid hardware and software solution. It can be noted that our implementation clearly outperforms the published computational times.

Table 5 Performance comparison with other works of the literature

Paper	Image resolution	Performance (ms)
[14]	128 × 128	45.40
[15]	256 × 256	50.00
[3]	320 × 240	45.45
Our work	320 × 240	9.596

One of our major contributions is the proposal of a hardware architecture for the processing of ADAS vision applications. Not all the works of the literature develop their own ad hoc hardware, normally commercial development boards are used, which are of course not valid for use in a vehicle. Hsiao and Yeh [14] mounted their design on a prototyping board, which is not valid for use in the real environment either. Furthermore, they do not offer any value of board's size, which is an important parameter. Anders et al. [3] seem to develop a board, but they do not offer enough data for comparison. The board presented in [31] measures 53mm × 33 mm, which is smaller than our board. However, they do not offer any performance measure, so it cannot be compared with us fairly. Furthermore, as it is said in Sect. 1, they use ASIC, which has an important drawback: it cannot be reconfigured.

To sum-up it can be said that our system enhances the state of the art in terms of performance and it offers a novel hardware architecture for an ad hoc ADAS vision processing hardware.

5 Discussion

This paper presents a reconfigurable embedded vision system for ADAS applications in smart cities. The system has been designed using a hardware/software codesign methodology. Codesign techniques are useful to coordinate the design tasks of hardware and software teams as well as to obtain an efficient design chain. Traditionally, software teams started developing once the hardware was ready. Instead of this, codesign promotes a concurrent hardware and software development, what can shorten the total development time. This is specially important in the automotive industry, where the design cycles are becoming more and more short.

One of the most crucial stages in the system design is the system platform selection. To choose the appropriated platform, several important aspects such as costs, power consumption, computational performance, etc. should be taken into account. The proposed embedded system uses a Zynq 7000 SoC, which is formed by an ARM Cortex-A9 based processing system (PS) and 28 nm programmable logic (PL). This type of platforms, such as Xilinx's Zynq

7000, Altera's Arria V SoC or Cyclone V SoC, has many advantages. Firstly, their high interconnectivity bandwidth allows levels of performance that two-chip solutions cannot match. Furthermore, as they have a higher integrated architecture, the overall power is reduced. Specifically, the Zynq-7000 EPP offer several techniques to lower the system power consumption. This platform allows us to switch off the programmable logic or disable the PL clock. These techniques can reduce significantly the dynamic power consumption. Additionally, an SoC solution achieves to reduce the overall system costs and the platform size since it integrates the necessary processing parts into a chip including also their interfaces. Therefore, the system needs less external components and connections facilitating the production stage.

Its reconfiguration capability is one of the most important points of the presented embedded system. The processor in the PS boots first, and configures the PL. The software running in the PS can also be updated at any moment via Ethernet. If there is a design bug, this system permits to fix it after the product launch. Not only this, the whole application can be dynamically programmed. For example, we can use a different algorithm when we are driving in a city or when we are in a rural road, or between daytime and night-time. Instead of having all the algorithms implemented in a custom hardware, the PS can select the most appropriate algorithm, reconfigure the PL and continue processing. An external device can also reconfigure the application running in the PS to adapt it to the new driving conditions.

Such a dynamic reconfiguration capability makes the system highly flexible. The possibilities are endless. For example, we can configure the system depending on the city where we are driving. Using wireless connection, we can receive information from the city we are entering and reconfigure the system to adapt it to the new scenario. In short, the proposed reconfigurable system can help to develop smart cities making vehicles smarter.

The results obtained in the case study show that the proposed system is suitable for implementing driver assistance vision applications. The PL can speed up significantly pixel-level operations, while the PS is in charge of the serial processing. A total computational time smaller than 10 ms was obtained for the LDWS algorithm, achieving the target time budget.

6 Conclusions

There are many options to implement a computer vision ADAS algorithm into an embedded system. Microprocessors and DSPs provide fast developing cycles and a low level of design complexity, but they are not enough for a real-time

performance for the most demanding applications. On the other hand, a hardware implementation in an FPGA can improve performance and energy efficiency, but at the same time the development cycle increases dramatically. System on Chips offer a trade-off between both worlds. Their architectures permit to implement custom logic in the PL and custom software in the PS. As the whole platform is integrated in the same chip, it has a better performance and power efficiency than traditional two-chip systems.

This paper presents an embedded vision system based on an SoC platform over our own tailored PCB. The system meets the requirements of the ADAS. The computational performance of the proposed embedded system improves the works found in the literature. Additionally, its reconfiguration capabilities increase significantly the application possibilities. In the context of smart cities, we can obtain smarter vehicles that can adapt dynamically to their environment. These smart vehicles would be able to improve our driving experience, minimise traffic accidents or prevent traffic congestion. In other words, they could improve our life quality.

Acknowledgments This work has been partially supported by the program ETORGAI 2011–2013 of the Basque Government under project IEB11. This work has been possible thanks to the cooperation with Datik-Irizar Group for their support in the installation, integration and testing stages of the project.

References

1. Aggarwal, A.: Embedded vision system (EVS). In: 2008 IEEE/ASME International Conference on mechatronic and embedded systems and applications. MESA 2008, pp. 618–621 (2008)
2. An, X., Shang, E., Song, J., Li, J., He, H.: Real-time lane departure warning system based on a single FPGA. *EURASIP J. Image Video Process.* **1**, 1–18 (2013)
3. Anders, J., Mefenza, M., Bobda, C., Yonga, F., Aklah, Z., Gunn, K.: A hardware/software prototyping system for driving assistance investigations. *J. Real Time Image Process.* (2013). doi:[10.1007/s11554-013-0351-4](https://doi.org/10.1007/s11554-013-0351-4)
4. Aurigi, A.: Making the digital city: the early shaping of urban Internet space. Ashgate Publishing, Farnborough (2005)
5. Batty, M., Axhausen, K., Giannotti, F., Pozdnoukhov, A., Bazani, A., Wachowicz, M., Ouzounis, G., Portugali, Y.: Smart cities of the future. *Eur. Phys. J. Spec. Top.* **214**(1), 481–518 (2012)
6. Buch, N., Velastin, S., Orwell, J.: A review of computer vision techniques for the analysis of urban traffic. *IEEE Trans. Intell. Transp. Syst.* **12**(3), 920–939 (2011)
7. Chakraborty, S., Lukasiewicz, M., Buckl, C., Fahmy, S., Chang, N., Park, S., Kim, Y., Leteinturier, P., Adlkofer, H.: Embedded systems and software challenges in electric vehicles. In: Proceedings of the Conference on design, automation and test in Europe, DATE '12, pp. 424–429 (2012)
8. Chang, S.L., Chen, L.S., Chung, Y.C., Chen, S.W.: Automatic license plate recognition. *IEEE Trans. Intell. Transp. Syst.* **5**(1), 42–53 (2004)
9. Darouich, M., Guyetant, S., Lavenier, D.: A reconfigurable disparity engine for stereovision in advanced driver assistance

- systems. In: Sirisuk, P., Morgan, F., El-Ghazawi, T., Amano, H. (eds.) *Reconfigurable Computing: Architectures, Tools and Applications*. Lecture Notes in Computer Science, vol. 5992, pp. 306–317. Springer, Berlin (2010)
10. Dong, Y., Hu, Z., Uchimura, K., Murayama, N.: Driver inattention monitoring system for intelligent vehicles: A review. *IEEE Trans. Intell. Transp. Syst.* **12**(2), 596–614 (2011)
 11. Geronimo, D., Lopez, A., Sappa, A., Graf, T.: Survey of pedestrian detection for advanced driver assistance systems. *IEEE Trans. Pattern Anal. Mach. Intell.* **32**(7), 1239–1258 (2010)
 12. Harrison, C., Eckman, B., Hamilton, R., Hartswick, P., Kalaganam, J., Paraszczak, J., Williams, P.: Foundations for smarter cities. *IBM J. Res. Dev.* **54**(4), 1–16 (2010)
 13. Hiraiwa, J., Amano, H.: An FPGA implementation of reconfigurable real-time vision architecture. In: 27th International Conference on advanced information networking and applications Workshops (WAINA), pp. 150–155 (2013)
 14. 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*. VTC 2006-Spring 6:2982–2986 (2006)
 15. 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 2009, pp. 319–323 (2009)
 16. Lee, S., Son, H., Choi, J.C., Min, K.: High-performance hog feature extractor circuit for driver assistance system. In: 2013 IEEE International Conference on consumer electronics (ICCE), pp. 338–339 (2013)
 17. Lin, H.Y., Chen, L.Q., Lin, Y.H., Yu, M.S.: Lane departure and front collision warning using a single camera. In: 2012 International Symposium on intelligent signal processing and communications systems (ISPACS), pp. 64–69 (2012)
 18. Malinowski, A., Yu, H.: Comparison of embedded system design for industrial applications. *IEEE Trans. Ind. Inform.* **7**(2), 244–254 (2011)
 19. Mielke, M., Schafer, A., Bruck, R.: Asic implementation of a gaussian pyramid for use in autonomous mobile robotics. In: 2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 1–4 (2011)
 20. Nieto, M., Cortés, A., Otaegui, O., Arróspide, J., Salgado, L.: Real-time lane tracking using rao-blackwellized particle filter. *J. Real Time Image Process.* (2012). doi:10.1007/s11554-012-0315-0
 21. Pedre, S., Krajník, T., Todorovich, E., Borensztein, P.: Accelerating embedded image processing for real time: a case study. *J. Real Time Image Process.* (2013). doi:10.1007/s11554-013-0353-2
 22. Samarawickrama, M., Pasqual, A., Rodrigo, R.: FPGA-based compact and flexible architecture for real-time embedded vision systems. In: 2009 International Conference on industrial and information systems (ICIIS), pp. 337–342 (2009)
 23. Sánchez-Oro, J., Fernández-López, D., Cabido, R., Montemayor, A.S., Pantrigo, J.J.: Urban traffic surveillance in smart cities using radar images. In: Ferrández Vicente, J.M., Álvarez Sánchez, J.R., Paz López, F., Toledo Moreo, F.J. (eds) *Natural and artificial computation in engineering and medical applications*, lecture notes in computer science, vol. 7931. Springer, Berlin, pp. 296–305 (2013)
 24. Sanders, L.: Secure boot of Zynq-7000 All programmable SoC. Application note XAPP1175 (v1.0), Xilinx (2013)
 25. Sandino, D., Matey, L.M., Vélez, G.: Design thinking methodology for the design of interactive real-time applications. In: Marcus, A. (ed.) *Design, user experience, and usability*. Design philosophy, methods, and tools, lecture notes in computer science, vol. 8012. Springer, Berlin, pp. 583–592 (2013)
 26. Schneiderman, R.: Car makers see opportunities in infotainment, driver-assistance systems [special reports]. *IEEE Signal Process. Mag.* **30**(1), 11–15 (2013)
 27. Shaout, A., Colella, D., Awad, S.: Advanced driver assistance systems - past, present and future. In: 2011 Seventh International Computer Engineering Conference (ICENCO), pp. 72–82 (2011)
 28. Shreejith, S., Fahmy, S., Lukasiwycz, M.: Reconfigurable computing in next-generation automotive networks. *IEEE Embed. Syst. Lett.* **5**(1), 12–15 (2013)
 29. Souani, C., Faiedh, H., Besbes, K.: Efficient algorithm for automatic road sign recognition and its hardware implementation. *J. Real Time Image Process.* **9**(1), 79–93 (2014). doi:10.1007/s11554-013-0348-z
 30. Stein, F.: The challenge of putting vision algorithms into a car. In: 2012 IEEE Computer Society Conference on computer vision and pattern recognition workshops (CVPRW), pp. 89–94 (2012)
 31. Stein, G., 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*, 2005. CVPR workshops, pp. 130–130 (2005)
 32. Teich, J.: Hardware/software codesign: the past, the present, and predicting the future. *Proc. IEEE* **100**(Special Centennial Issue):1411–1430 (2012)
 33. Toral, S., Barrero, F., Vargas, M.: Development of an embedded vision based vehicle detection system using an ARM video processor. In: 11th International IEEE Conference on intelligent transportation systems, 2008. ITSC 2008, pp. 292–297 (2008)
 34. Turturici, M., Saponara, S., Fanucci, L., Franchi, E.: Low-power DSP system for real-time correction of fish-eye cameras in automotive driver assistance applications. *J. Real Time Image Process.* (2013). doi:10.1007/s11554-013-0330-9
 35. Vicomtech-IK4: Viulib: Computer Vision SDK. URL <http://www.vicomtech.org/viulib/> (2013)
 36. Wójcikowski, M., Zaglewski, R., Pankiewicz, B.: FPGA-based real-time implementation of detection algorithm for automatic traffic surveillance sensor network. *J. Signal Process. Syst.* **68**(1), 1–18 (2012)
 37. Wu, B.F., Huang, H.Y., Chen, C.J., Chen, Y.H., Chang, C.W., Chen, Y.L.: A vision-based blind spot warning system for daytime and nighttime driver assistance. *Comput. Electr. Eng.* **39**(3), 846–862 (2013)
- Gorka Velez** received his M.Sc degree in Electronic Engineering from the University of Mondragon (Spain) in 2007, and his Ph.D. from the University of Navarra (Spain) in 2012. He currently works as a researcher of the Intelligent Transportation Systems and Engineering Area of Vicomtech-IK4 research center. His research interests include real-time systems, design methodologies and intelligent transportation systems.
- Ainhoa Cortés** received her degree in Electrical, Electronic and Control Engineering from the University of Navarra, Spain, in 2004, and her Ph.D. in 2007. She is currently working in the Electronics and Communications department of CEIT-IK4 (San Sebastian, Spain) and is a lecturer at the University of Navarra (Spain). Her research interests are in the area of the design and implementation of digital signal processing algorithms.
- Marcos Nieto** received the Ingeniero de Telecomunicación degree in 2005 and the Ph.D. degree in 2010 both from the E.T.S.Ing. Telecomunicación (ETSIT) of the Universidad Politécnica de Madrid (UPM), Spain. He currently works in the Vicomtech-IK4 research center, in which he is working in the area of Intelligent Transportation

Systems and Engineering. His actual research interests include the use of optimization methods for probabilistic models in computer vision as well as the design of H.264/AVC video codecs.

Igone Vélez received her M.Sc. Degree in Electronic Engineering from the University of Navarra in 2000, and the Ph.D. in 2005. She has been a research staff member at the Electronic and Communications department of CEIT-IK4 (Spain) since 2000 and an associate professor at the University of Navarra (Spain) since 2005. Her

research interests include design methodologies, digital signal processing and digital hardware design.

Oihana Otaegui received her M.S. and Ph.D. degrees in Electronic Engineering from the Tecnun, University of Navarra, Donostia–San Sebastian, Spain in 1999 and 2005 respectively. She is currently the head of the ITS and Engineering Area of Vicomtech-IK4, Donostia–San Sebastian, Spain. Her research interests include satellite navigation and transport fields.