HD Number Plate Localization and Character Segmentation on the Zynq Heterogeneous SoC

Ali Al-Zawqari*[1], Omar Hommos*[1], Abdulhadi Al-Qahtani[1], Ali A. H. Farhat[1], Faycal Bensaali[1], Xiaojun Zhai[2] and Abbes Amira[1]

[1] *College of Engineering*, Qatar University, Doha, Qatar
[2] *Department of Computing and Mathematics*, University of Derby, Derby, UK
*Equal contribution to the work, {Ali.M.Al-Zawqari, Omar.G.Hommos}@ieee.org

***Abstract*— Automatic Number Plate Recognition (ANPR) systems have become widely used in safety, security, and commercial aspects. A typical ANPR system consists of three main stages: Number Plate Localization (NPL), Character Segmentation (CS), and Optical Character Recognition (OCR). In recent years, to provide a better recognition rate, High Definition (HD) cameras have started to be used. However, most known techniques for Standard Definition (SD) are not suitable for real-time HD image processing due to the computationally intensive cost of processing several-folds more of image pixels, particularly in the NPL stage. In this paper, algorithms suitable for hardware implementation for NPL and CS stages of an HD ANPR system are presented. Software implementation of the algorithms was carried on as a proof of concept, followed by hardware implementation on a heterogeneous System on Chip (SoC) device that contains an ARM processor and a Field Programmable Gate Array (FPGA). Heterogeneous implementation of these stages has shown that this HD NPL algorithm can localize a number plate in 16.17 ms, with a success rate of 98.0%. The CS algorithm can then segment the detected plate in 0.59 ms, with a success rate of 99.05%. Both stages utilize only 21% of the available on-chip configurable logic blocks.**

Keywords*— Number Plate Localization; Automatic Number Plate Recognition Systems; Character Segmentation; Heterogeneous SoC; High-Level Synthesis.

I. INTRODUCTION

Intelligent Transportation Systems (ITS) play a key-role in modern-day infrastructure development, where government institutions and commercial organizations seek the use of advanced technology to improve safety, comfort, and quality of transportation systems. Automatic Number Plate Recognition (ANPR) systems are an essential part of ITS, and a critical component in smart cities, especially for infrastructure applications such as access control, motorway road tolling, border control, identification of stolen cars, and traffic law enforcement. Its proven effectiveness in such applications made it a well-researched and a widely deployed solution for similar use cases.

Generally, an ANPR system consists of three main stages: Number Plate Localization (NPL), Character Segmentation (CS), and Optical Character Recognition (OCR). The NPL stage takes the input car image, localizes the NP within it, and sends it to the CS stage where the plate characters are segmented. The segmented number characters are then passed to the OCR stage to be recognized, where the final output is encoded text.

Recently, research in ANPR systems has moved to the use of High Definition (HD) images instead of Standard Definition (SD) images, since using HD images can improve the recognition rate of the system because it offers higher quality images. In addition, HD cameras cover wider street area in comparison with SD cameras, so a single camera can be used to cover several lanes at the same time. These advantages come with the drop back of having an increasingly larger number of pixels to process, which leads to more processing time. Thus, HD images processing poses a challenge to meeting real-time processing requirements of ANPR systems [1]. This challenge holds especially for the NPL stage, as it is the most time-consuming stage of the system [2].

Furthermore, CS stage is important since degraded NP images are still a problem that increases segmentation errors in ANPR systems. This impact can be effectively reduced through combining proper methodologies. Moreover, correct segmentation can ease things in the OCR stage, leading to higher recognition rates in the OCR, and a better overall performance for the full system. A good CS algorithm must be able to segment characters out of reasonably degraded NPs, NPs that are under different illumination conditions, and rotated NPs that might get extracted from the images in the NPL stage.

The increasing complexity of ANPR algorithms and the high computational cost of image processing operations motivated the use of high performance workstations and super-computers as the target hardware platforms for such systems, especially when real-time processing is needed [3]. The need to use these hardware platforms leads to other issues related to high power consumption, cost and compactness [4]. This motivated researchers to look for other platforms for ANPR implementation. These platforms include Field Programmable Gate Arrays (FPGAs), Digital Signal Processors (DSPs), hybrids of these different platforms, and heterogeneous System on Chip (SoC) devices that host dissimilar system components on a single chip. The mentioned platforms are considered as a low-cost alternative for accelerating such computationally intensive task [5].

In this paper, the proposed work focuses on the development and the hardware implementation of HD NPL and CS stages, with three main considerations in mind: real-time processing, high success rate, and low hardware utilization. MATLAB implementation of the proposed algorithm was used as a proof of concept prior to hardware implementation.

This is part of a research work that aims at implementing the whole HD ANPR system on a single chip. The used hardware device is a heterogeneous SoC named Xilinx® Zynq-7000 All Programmable SoC. It contains both a Programmable Logic (PL) that is an FPGA, which is used to describe hardware architectures to accelerate computationally intensive blocks, and a Processing System (PS) that is an ARM® Cortex-A9 based application processor, which is used to interact with peripherals, run embedded operating systems, or run specific system blocks [6]. This hardware platform will allow doing all the computations on a single chip that is cheap, power efficient, and compact in size, thus can be embedded in the camera itself, instead of having a separate processing unit.

The remaining sections of this paper are organized as follows: Section 2 discusses related work. Proposed algorithms for HD NPL and CS are presented in section 3. Section 4 describes the hardware block design and its implementation on the Zynq SoC. Implementation results and discussion are presented in section 5. Section 6 concludes the paper.

II. Related Work

This section begins with a discussion of previous NPL algorithms and implementations, followed by a similar discussion for the CS stage. The comparison of related work with this current work is presented in section VI.

As the field of FPGAs and DSPs has been evolving lately, researchers and engineers have been motivated to design, test, and implement their ANPR systems on such devices. Researchers in [5] were able to design an NPL system on hybrid DSP/FPGA platform. The DSP is the main unit where the processing was done, and the FPGA was used to buffer the video frames that are acquired from a video processor in the camera. This NPL uses a Viola Jones detector, their results take 140 ms to process one frame of size 352×288, and out of 100 NPs; 96% were recognized.

Another group of researchers were able to develop an ANPR system on a DSP platform that processes a video stream in real time in [7]. Their system consists of three main stages: NPL, CS, and OCR where A Viola Jones detector is used to detect NPs in the NPL stage. In addition, a tracker process is used to track and relate the detected plates between the different frames. Afterwards, the detected plates are passed to the recognition stage where the characters will be segmented first and recognized later by a classification process. Historical data is used to confirm the recognition process in a post-processing unit. The resolution of each frame is 352×288 and each frame is processed on its own. The obtained result is transmitted to the user in addition to a compressed image. The Viola Jones and support vector machine classifiers were trained on a general purpose computer before applying them on the DSP platform. Two data sets were used. The former consists of 260 number plate images which is used for training purposes. The latter is a group of frames that are extracted from a video that were used to test the performance of the system. The system takes around 7.3 ms to localize an NP and 52.11 ms to process one frame on average.

In [8], an FPGA-based NPL system was developed to detect NPs and measure their size. The size is used to calculate the distance between two vehicles in order to warn the drivers of the danger of collision. First, the algorithm converts a 24-bit RGB image to an 8-bit grayscale image, and then it converts the image to a binary image based on pixels' classification. Afterwards, each region in the image that has neighboring pixels with the same colors labelled. Then, based on four conditions, the NP candidate regions in the image are located. Later, three more conditions are used to select the NP region. The image resolution is 256×256. The image is divided into 16 smaller images, each is 16×256, and the segments are processed in parallel. The system was simulated in software to prove the concept. The software implementation takes around 38.52 ms to process one image, whereas the hardware takes 9.25 ms only. The detection rate was found to be 87.7% and 84.1% during the day and night times respectively.

In [9], an improved algorithm for NPL stage was developed and implemented on FPGA. The NPL proposed algorithm consists of two main sub-stages: Plate feature extraction, and selection of candidate plate regions. The first stage is based on morphological operations, where Connected Component Analysis (CCA) is used to locate the plate in the image for the second stage. The algorithm was proposed to process SD images with a resolution of 640×480. Two different database sets were used for testing, one from the UK with 1000 images and another from Greece with 307 images. The UK database was divided into six different sets, whereas the Greek database was divided into three. Those sample sets differ in terms of distance and illumination conditions. First, NPL was implemented in MATLAB and results showed a success rate of 97.9% and 98% using the UK and Greece databases respectively. Thereafter, it was implemented on FPGA, showing a success rate of 97.7% and 98% using the UK and Greece databases respectively. About 33% of the FPGA resources were required to implement this proposed architecture, while being able to localize an NP in 4.7 ms.

The most recent work on NPL presented in [10] uses a two-stage approach. First, a weak Sparse Network of Winnows (SNoW) learning architecture extracts a set of candidate regions that are then passed to a strong Convolutional Neural Network (CNN) classifier that determines the number plate. An interesting take on this work is that it further includes a classifier that analysis failed images (where no plate could be found) to determine reason of failure; whether NP is too bright, NP is too dark, NP not present, or vehicle not present. Software implementation only of the proposed algorithm has been performed. No hardware implementation is presented as

running complicated networks like large CNNs on hardware is resource demanding, and is still in early research stages.

The CS stage, due to its importance as part of an ANPR system, has been a topic of research for several years now. Researchers aim to develop efficient segmentation algorithms in order to increase the segmentation rate and processing time. CS is an important stage where each character is fully isolated before applying OCR. In fact, most errors in any ANPR system are not due to NPL or OCR stages, but due to character segmentation [11]. In recent years, improved or modified CS techniques have been developed in order to overcome the noisy NP images (e.g. uneven illumination, inclined NP, and connected plate characters) [12]. The methods that are commonly used in the CS stage are divided into three main categories: projections and binary algorithms, contours tracking algorithms, and classifiers based algorithms [13].

The most commonly used CS algorithm is the one based on the projections of NP pixels [12]. The idea of this algorithm is to obtain the vertical and horizontal projections, and based on the projection histograms, the characters can be segmented. The proposed algorithm in [12] was implemented using the Mentor Graphics RC240 FPGA development board equipped with a 4M-Gate Xilinx Virtex-4 LX40. It is capable of processing one image in 0.2 – 1.4 ms with a success rate of 97.7% , and was tested using a database of 1,000 binary UK NPs with varying resolutions. The work in [14] presents a video processing methodology for an FPGA-based ANPR system. The pixel projection method was applied in the CS stage, but to improve the segmentation results, an adjustment to the critical point finding method was carried out. The updated CS algorithm works directly with the OCR stage, and relies on continuous sweeping of the threshold point until enough NP character regions are segmented (7 characters for that NP).

Another method for CS that falls under the same category is CCA. This method was used in [15]. It is based on a set of geometric measurements that need to be calculated and compared with some conditions to extract the characters. This way, characters can be extracted even if the NP is rotated, but it requires each character to be fully connected and isolated from all other characters. CCA method must be applied only to the binary images. Therefore, an NP binarization stage is required.

The work in [16] presents a combination of the projection method and CCA. This combination has the ability of segmenting characters that consist of multiple line segments, e.g. Chinese or Korean alphabets. It uses CCA as a primary way of segmenting, and pixel projection for second-stage processing, in case a failure in segmentation occurred.

The second category of CS algorithms is contour tracking. In [17], the contour curve of each character was extracted into four and eight directions by using 2×2 and 3×3 masks respectively. The NP will be then divided into two parts using density-indicating histogram for y-axis direction. This algorithm is capable of segmenting broken characters.

The third category for CS is based on classifier networks. The method proposed in [18] was developed for CS in NP video sequences. This algorithm is based on the extraction of characters as a Markov Random Field (MRF), where previous knowledge of NP is used to maximize a posteriori probability. The method presented in [13] uses prior knowledge of the number of characters in the NP with equally width segmentation. Nevertheless, the width of the NP characters is normally unknown as the characters can vary from an NP to another and inclined NPs can affect the width of the characters. The work in [19] implements a CS algorithm on TI C64 DSP. The characters on target NPs are black over a white background. The method relies on region-growing, seeded with lowest intensity values. The region descriptor of histogram, compactness, and other features are incrementally computed, and are then classified using a Support Vector Machine.

III. Proposed HD NPL Algorithm

As discussed in the previous section, many NPL algorithms have been proposed. However, moving from SD to HD images severely affects the time performance of most of these algorithms, as a huge amount of data must be processed while those algorithms are not tailored to deal with this large amount of pixels. The used algorithm for this HD NPL stage has been proposed by the authors in [20]. Fig. 1 shows the block diagram of the algorithm. It is based on HD-to-SD image resize, SD NPL, and SD-to-HD coordinate transformation.

First of all, an HD-to-SD image resize operation is used to reduce the amount of pixels in the image. The selected HD resolution, which is 960×720, has 2.5 times more pixels than the SD image with the resolution of 640×480. This substantial decrease in pixels increases processing time and decreases the hardware utilization on the PL. Afterwards, an SD NPL algorithm that was developed with hardware compatibility in mind is used to localize the plate. After NP localization has been achieved with SD image, coordinates transformation is done to specify its location in the HD image.

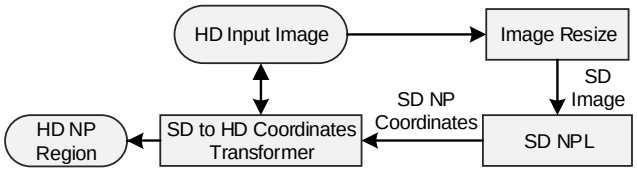


Fig. 1. HD NPL block diagram [20]

As for the SD NPL algorithm, it is mainly based on morphological operations and CCA [20], and is illustrated in Fig. 2. The image is first converted into grayscale to reduce the required amount of processed information. Morphological operations are then used for feature extraction. They are applied as a light-weight alternative to edge detectors, and work as follows: first, an image Open operation is performed on the grayscale image, using a structural element (SE) with a shape similar to the number plate. An SE of size 3×30 was found to be satisfactory. The result is a blurred grayscale image, with maximum blur occurring in the NP region (as the SE shape depicts that of the NP). Subtracting the blurred image from the grayscale image will result in a new grayscale image that contains high white-edges density around the plate region. This image is called the Highlighted Plate Region image. It is then binarized at using fixed threshold to obtain a binary image. An image Open operation with a small SE is used to clean up noise pixels. A following image Close operation is then applied to fuse the edges and create an image with white

components (sometimes called blobs) on a black background. This image is passed to a CCA algorithm that labels every white component using 4-connectivity method.

For each labelled component, specific geometrical properties are calculated, i.e. width, height, width/height ratio, and the number of white pixels of a component normalized to its area (width × height). Those are then compared against some pre-specified geometrical conditions that enable selecting the character components out of all white image components. When the NP location is determined, and using simple coordinate transformation operations, the location of the NP in the HD image can be specified. The HD NP image can then be passed to the following stages [20].

A modification to the algorithm has been applied and explained in the following section. This modified algorithm was implemented and tested using MATLAB, as a proof of concept, with 958 front-view HD images. Results have shown a success rate of 98% and can detect an HD NP in 32.48 ms.

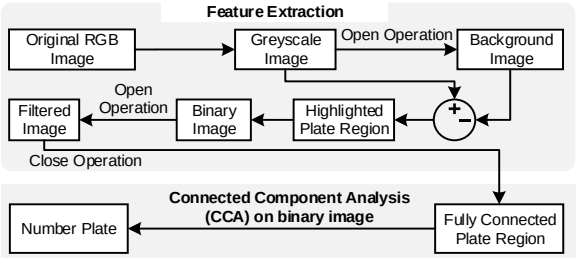


Fig. 2. Proposed SD NPL Algorithm [20]

IV. Proposed Character Segmentation Algorithm

As discussed previously, a good segmentation algorithm is essential for the next OCR stage, and thus for the overall performance of the system. Furthermore, NPs taken from HD images would have a larger area, thus a good algorithm needs to take advantage of that to increase the recognition rate without affecting the time performance. The algorithm for this hardware CS implementation has been proposed by the authors in [20]. Some modifications have been added to make it more hardware-compatible. Fig. 3 shows the block diagram of the proposed algorithm. First of all, a mean-based adaptive binarization process is applied to the input NP image. This process adaptively binarizes the image based on the calculated mean, and is explained thoroughly in a moment. Afterwards, an 8-connectivity CCA is applied to extract the character components out of all available white components in the binarized NP. Image close operations are applied throughout the process to connect any broken NP characters, and to enhance the segmented characters before sending them to the OCR stage. Fig. 4 shows an example of an input NP, the adaptive binarization output, and segmented NP characters.

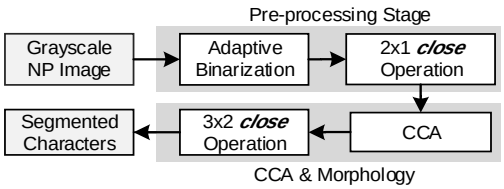


Fig. 3. Character Segmentation Block Diagram [20]

Fig. 4. A Qatari NP with the binarization process output and its fully segmented characters.

The adaptive threshold operation allows better binarization of NPs that are under uneven illumination, in shadow, or extracted from images captured at night. This boosts CS stage performance significantly, especially when the complexity of real-world car images is taken into consideration.

To perform adaptive binarization, a square window of size $w \times w$ scans the NP, where each pixel becomes the center of that window once. At that instance, the mean of the pixels that fall under the window is calculated, the local threshold $T_B(x, y)$ for that point is then obtained by:

$$T_B(x, y) = f_{mean}(x, y) - \Delta t \quad (1)$$

Where Δt is the threshold offset, which is used for threshold adjustment. This threshold is then used to binarize image pixels that belong to the window. In this algorithm, w and Δt were adjusted based on experimental results. A value of 10 for Δt was found to have the best experimental results, while a window size $w = 9$ was found to be good [20].

Following adaptive binarization, a 2×1 image close operation is used to fuse any disconnected NP characters, e.g. characters that are split because the NP is damaged or because of a shadow line over the NP in the image. Results are shown in Fig. 5.

Fig. 5. Before (right) and after (left) applying a 2×1 close operation.

Afterwards, CCA is computed. First, an 8-connectivity method is used to group all the white pixels in the previous binary NP image, meaning that when two adjoining pixels are high "1" (whether they are connected horizontally, vertically, or diagonally), they are a part of the same white component. All components are labelled. A loop afterwards checks the labelled components one by one, calculates some specific geometrical parameters for each component like the width, height, area, and the perimeter (the length of the region boundary) of that component, and compares it to a set of defined geometrical conditions. If the component meets those conditions, then it is a number character. Number characters can be enhanced furthermore, depending on the utilized OCR stage, by performing a 3×2 image close operation. This fills any holes inside the character and enhances the general shape of the segmented character. Results are shown in Fig. 6.

Fig. 6. Before (left) and after (right) applying 3×2 close operation.

More elaborate description and analysis of the algorithm, along with many more descriptive figures, can be found in [20]. A review and more details about the used image processing operations can be found in [21, 22].

V. Hardware Block Design

The Zedboard [23] was the selected hardware development platform for HD NPL. This platform is equipped with the Zynq-7000 All Programmable SoC, a heterogeneous chip that combines the software programmability of a dual core ARM Cortex-A9 based PS, with the hardware programmability of the PL that is an FPGA [6]. This enables exploiting parallelism in the PL fabric for computationally intensive tasks, while the PS is busy doing other simple tasks. Furthermore, many systems that utilize FPGAs require implementing a soft-core processor in the fabric for specific tasks, e.g. communicating with peripherals, or tasks with a high degree of flexibility [24]. Hard-core processors can run at a frequency higher than that of the PL fabric, so having it besides the PL on a single chip with the ability to exchange the data between those two components is beneficial in terms of speed. In addition, it facilitates using some tools that already exist for hard-core processors, such as embedded operating systems or some other design tools.

To identify what to run on the PL or on the PS, the full HD NPL and CS stages will be run on the PS in order to test and time analyze different blocks of the algorithm. The most time-consuming operations will then be moved to PL.

The OpenCV library was used to run the stages on the PS. This library functions are highly optimized for real-time applications. To run the OpenCV library, an operating system has to be installed on the PS, as it requires some high level system functions. The OpenCV library was compiled keeping in mind that the PS hardware includes NEON media-processing engine, and a Vector Floating-Point Unit (VFPU).

Linaro [25], a distribution that is based on Ubuntu Linux-based operating system, and ported especially for ARM embedded system applications, was installed on the Zedboard. In Table I, the results for timing calculations for different sections of the stages running on Linaro, and a comparison of results with the same sections running on Ubuntu. Note that Ubuntu was running on a virtual machine on a desktop computer with Windows operating system. The settings of the virtual machine were chosen to allow a maximum of 1 GHz of the processing power, similar to the CPU frequency of the PS.

TABLE I. PS TIME PERFORMANCE ESTIMATES

Stage	Function	Ubuntu (ms)	Linaro (ms)
HD NPL	**Resize**	16.87	20.23
	Morphorigcal Opereations	6.27	217.42
	CCA	4.49	8.08
CS	**Adaptive Threshold**	0.19	0.64
	2 × 1 Image Close	0.03	0.23
	CCA	0.14	0.51
	Total	0.36	1.38

HD NPL results show that image resizing and morphological operations are time consuming when they run on the PS, which causes a serious degradation in the overall performance. Both operations can be performed at a much faster rate when implemented in the fabric by exploiting the hardware flexibility of the PL, and the ability to run operations in a pipelined manner. Based on this, those operations will be moved to the PL, while the CCA will remain on the PS. This takes into consideration the heterogeneous properties of this platform, as it allows accelerating the slow blocks, while other fast blocks of the system can be run and implemented on the PS using already available functions and tools.

CS results show that the adaptive threshold operation is relatively time consuming when it is run on the PS, so it has been moved to the PL, along with the image close operation that can be easily implemented there. Both operations can be performed at a much faster rate when implemented in the fabric by exploiting the hardware flexibility of the PL, and the ability to run operations in a pipelined and parallel manner. The CCA will remain on the PS. This takes into consideration the heterogeneous properties of this platform, as it allows accelerating the slow blocks, while other fast blocks of the system can be run and implemented on the PS using already available functions and tools.

Fig. 7 shows the proposed hardware architecture for both stages. PL blocks design will be discussed in the coming sub-sections.

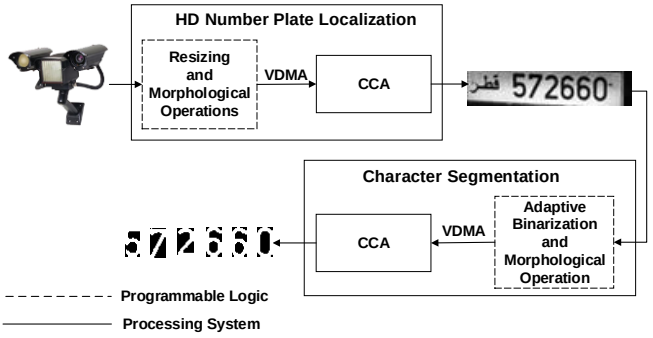


Fig. 7. Hardware block diagram

A. Interconnecting Hardware Blocks

In the Zynq SoC, AMBA® ARM AXI based protocols are used to interconnect the PS and the PL sides. It specifically uses the second version of AXI, AXI4. Direct Memory Access (DMA) protocol allows using the shared memory between the PS and the PL to move data back and forth. The Video Direct Memory Access (VDMA) engine provides a high-bandwidth solution for transferring large data such as videos or images, and is widely used in implementing image processing blocks or data filters [26]. This will be used to connect the PL and the PS blocks of this system. AXI VDMA latency does not exceed several cycles on its read and write channels. For a data width of 32 bits, VDMA throughput is equal to 96 frames per second [26]. This measure was done using standard 960×720 HD frames on hardware. Based on this, transfer speed is about 6.3

Megapixels/s, and about 10.4 ms to transfer a full HD frame from memory to the PL or PS, or vice-versa. The Pipelined hardware design allows the processing to start once a pixel arrives, and owing to the dual-port memory available on the board, the transfer destination will not have to wait for the full frame to arrive. This will save on these 10.4 ms, and the memory needed to buffer the frame. Therefore, data transfer time is neglected when calculating the overall time performance of this stage as a part of the full system.

B. HD NPL PL Hardware Block Design and Performance

This block includes image resizing and all morphological operations performed in HD NPL. MATLAB testing results have shown that this block consumes about 2/3 of the total HD NPL stage time [20], and as the HD NPL stage is the most time consuming stage of an ANPR system [2], the performance of this block is essential. Fig. 8 shows the block diagram of the operations inside this hardware block. It includes the operations used for feature extraction as in [20] with a modification for hardware purposes that will be explained later.

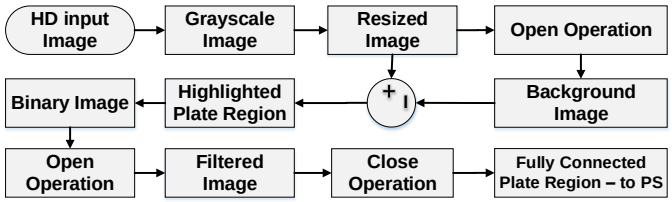


Fig. 8. HD NPL PL block

The proposed hardware block has been synthesized using Xilinx Vivado High-Level Synthesis (HLS), which facilitates HLS creation of PL blocks [27]. Several HLS libraries are provided by Xilinx, including Vivado HLS Video Library [28]. All functions needed in this block are included in the library. Vivado HLS, with the support of its libraries, alleviates most of the work needed to create highly-optimized FPGA RTL implementations. It also provides different optimization techniques that can be applied and tuned by the user to maximize the hardware performance. These techniques are applied and tuned through directive pragmas and include loop flattening, pipelining (to reduce initiation intervals), array mapping (combines multiple smaller arrays into a single large array to help reduce BRAM resources), dataflow, and much more [29]. Each optimization technique requires a set of rules to be applied in order to be used, like eliminating inter-loop data dependencies, or eliminating conditional executions, non-existence of feedback between tasks, and so on. Performing HLS will generate a final RTL implementation that follows criterion specified by the user on a high-level, without worrying about low-level details like FIFO, BRAM, and DSP architectures.

Dataflow directive was used to allow pipelining between functions inside this PL block. It enables task-level pipelining, allowing functions to overlap in their operation, and increasing concurrency of the Vivado generated RTL implementation. Fig. 9 shows an example of how the Dataflow directive pragma operates. Once an input pixel is processed by a function (e.g. grayscaling), the resultant output pixel is immediately passed to the next operation (e.g. resizing) for processing. Meaning that a pixel stream is always flowing through all blocks, where each two consecutive blocks are connected by what is called a hardware "channel" or "stream". In all, each block does not have to wait for the full image output from the previous block, but rather takes an output pixel from the previous block as its input as soon as it is ready. This increases the overall throughput of the design [29].

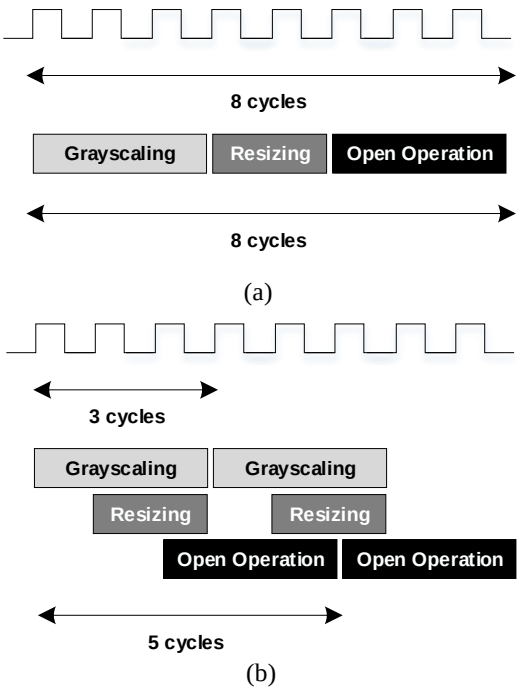


Fig. 9. Operation of Dataflow optimization (a) without dataflow pipelining (b) with dataflow pipelining. Illustrated function latencies are arbitrary.

When working with Dataflow, careful consideration of each function's initiation and latency should be taken. Vivado automatically optimizes function initiation intervals, such as each system block is initiated as fast as possible, and a block starts its operations as soon as its input data is available. However, each functional block has its own latency, which is the time needed to produce the first pixel output, upon receiving the first pixel input. This might create an issue if correct buffering is not applied at the branches of the block diagram. Looking back to Fig. 8, pixels flowing through the pixel "stream" that connects the Resized Image block to the Subtraction operation block should be buffered while the output of the Open operation is still not ready. This buffering is modelled by Vivado as a FIFO, and its depth (number of elements in the FIFO) should be at least equal to the latency (clock cycles spent until first output) of the Open operation block, such as all output elements from the Resize block get a buffer place until the Open block output starts flowing, and their consumption by the Subtraction operation starts. The Open block performs erosion followed by a dilation, where the

structural element of both operations has a height of 3, as seen in section III and in paper [20]. So, before an output could be released, about three lines of the image should be read by each of these operations. This gives an overall latency of $(3+3)\times 640 = 3840$ cycles for the Open operation block. The FIFO depth was set at 4000; rounding up for simplicity.

Table II shows the performance estimates of the optimized design, with a total of 8.09 ms at a clock period of 11.12 ns, in addition to showing the results for the non-optimized design, which is 7.3 times slower.

TABLE II. HD NPL PERFORMANCE ESTIMATES

Clock (ns)	Latency (clock cycles)	Total Time (ms)	Optimization Type
11.12	722,209	8.09	Dataflow
11.12	5,255,106	58.4	None

Table III shows detailed utilization estimates of this design, and with enough resources left to implement other stages of ANPR system on the PL. The DSP48E slices utilization was mainly due to the resize and the color conversion operations, with 16 slices used in the first, and the other 3 used in the latter. About 56% of the LUT utilization (4665 LUTs) was in the resize operation, as the Vivado HLS resize function uses bilinear interpolation, which is somehow costly in area in comparison with other simpler resize methods such as nearest neighborhood, i.e. the function uses many large size fixed-point operations which is also the reason behind its high contribution to the Flip-Flops (FF) and DSP48E slices consumption. BRAM utilization was mainly due to the morphological operations, with 3 BRAMs used for each of the six erosion or dilation operations. The other category includes 322 LUTs and 272 FFs used for interfacing i.e. I/O format conversion. A duplicate of the resized image stream must be conserved to be used later in image subtraction. The conservation was done by increasing the depth of the First-In First-Out (FIFO) used to model the stream. This high-depth FIFO consumed 177 LUTs, 70 FFs, and 2 BRAMs. A total of 1039 LUTs and 281 FFs were used for all implemented FIFOs.

In case dataflow optimization was not applied, each stage would have to finish processing a full image before passing output pixels to the next stage. This decreases performance by 7.3 times as seen in Table II; a quite inadequate result keeping in mind this only saves a fraction of fabric area. Utilization savings were in the form of two BRAMs, 1% in LUT count, and even less in FF count, which are the resources used to allow pipelined interconnects (the streams) between blocks. Utilization estimates are shown in Table IV. Note that even without dataflow optimization, thus with inter-block pipelining disabled, each block used by itself is highly optimized, i.e. optimization directives such as loop unroll are applied within the same block, as they were ready functions available from Xilinx in this way and used directly without any modifications.

TABLE III. DETAILED HD NPL UTILIZATION ESTIMATES– OPTIMIZED

Name	BRAM_18K	DSP48E	FF	LUT
Color Conversion	0	3	120	73
Resize	2	16	4,665	4,763
Erode, Dilate (6 total operations)	18	0	1,142	1,856
Image subtraction	0	0	47	88
Image Thresholding	0	0	39	55
Other	2	0	609	1,460
Total	22	19	6,622	8,295
Available	280	220	106,400	53,200
Utilization (%)	7	8	6	15

TABLE IV. HD NPL UTILIZATION ESTIMATES – WITHOUT OPTIMIZATION

Name	BRAM_18K	DSP48E	FF	LUT
Total	20	19	6,181	7,520
Available	280	220	106,400	53,200
Utilization (%)	7	8	5	14

Referring back to Fig. 7, performing image resize before color conversion, as was designed in [20] and shown in Fig. 2, will yield the same final output while keeping the total latency the same. Yet, this will cause a significant increase in PL area utilization, especially the DSP48E slices. This is because in comparison with grayscale images that have a single channel, color images have three channels. This raises the expected DSP48E slice utilization for image resizing to three times that of the grayscale image. Practical implementation showed that resize took 40 DSP slices, in comparison with 16 in the previous order, with an increase of 2.5 times.

Performance can be improved a little bit and utilization can be decreased vastly by implementing a nearest-neighborhood resize function that is simpler and uses less resources. However, with plenty of PL area still available, improvement in performance is not worth the engineering time at this stage.

C. CS PL Hardware Block Design and Performance

This block is essential in the CS stage, as it is what gives the CS algorithm the agility to process plates that are under various illumination conditions, and includes the adaptive threshold and the image close operations performed in CS. PS testing results have shown that this block consumes about 60% of the total CS stage time, so a good improvement in this stage, with plenty of PL area available to implement all system stages, the prospective improvement will improve the overall system performance by increasing the number of frames to be processed per second. Fig. 10 shows the block diagram of the operations inside this hardware block.

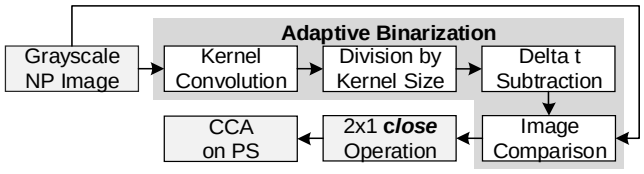


Fig. 10. CS PL block

Adaptive Threshold was implemented by applying an average filter to the image, subtracting the offset factor from all pixels, and then comparing the resultant image with the original image. Image averaging is done using two steps: first of all, a kernel of size $w \times w$ with all elements equal to 1 is convoluted with the image. Afterwards, each pixel of the output image stream is divided by w^2 using integer division which discards the fractional part (remainder) of the result. The resultant value will be compared to a grayscale image where pixels take integer values and by discarding the fraction it will have no effect on the resultant image, while achieving better performance and utilization, as no floating point operators are used. This allows the implementation of an averaging filter with minimum latency. The output stream is then processed by the next stage.

The proposed hardware block has been synthesized using Vivado HLS. All functions needed in this block are included in this Video HLS library. Dataflow directive was used to allow pipelining between functions inside this PL block, exactly as described in the previous subsection. Table V shows the performance estimates of this design, with a total of 0.08 ms at a clock period of 8.75 ns. It further shows a comparison of time performance results, with and without using the dataflow optimization directive. Table VI shows an overview of the utilization estimates of this stage for the optimized and non-optimized versions of the block. All results show high performance and low utilization for the hardware design.

TABLE V. CS PERFORMANCE ESTIMATES

Clock (ns)	Latency (clock cycles)	Total Time (ms)	Optimization Type
8.75 ns	9454	0.08	dataflow
8.75 ns	74421	0.65	none

TABLE VI. CS OVERVIEW OF UTILIZATION ESTIMATES

	Name	BRAM	DSP48E	FF	LUT
Dataflow	**Total**	12	0	1640	2706
	Available	280	220	106400	53200
	Utilization (%)	4	0	1	5
None	**Total**	8	0	1428	2227
	Available	280	220	106400	53200
	Utilization (%)	2	0	1	4

Detailed utilization estimates of the optimized version of this design are shown in Table VII. Kernel convolution was the most resource hungry operation, due to the relatively large size of the window i.e. $w = 8$. The remaining operations have low utilization in general. The "other" category includes four BRAMs used to buffer the grayscale image stream that was reserved to be used later in the image comparison operation. It also includes resources used for interfacing i.e. block I/O format conversion, and for FIFO and registers implementation.

As for the window size, it was found in [20] that $w = 9$ and $\Delta t = 10$ are the most fitting parameters for the current database. However, to reduce hardware utilization, the window size was set to $w = 8$ while the threshold adjustment constant was increased to $\Delta t = 15$. The final results are the same.

TABLE VII. CS DETAILED UTILIZATION ESTIMATES

Operation	BRAM_18K	DSP48E	FF	LUT
Kernel Convolution	8	0	1053	1658
Integer Division	0	0	24	37
Δt subtraction	0	0	41	73
Image Comparison	0	0	31	46
2x1 Close Operation	0	0	70	86
Other	4	0	421	806
Total	12	0	1640	2706

The output of this block is sent to the PS for the CCA. Afterwards, the segmented number characters may undergo the second morphological close operation if needed, before being passed to the OCR stage.

VI. IMPLEMENTATION RESULTS AND DISCUSSION

The performance of a stage as a part of the ANPR system can be divided into three categories: success rate, time performance, and overall hardware utilization.

For each of the NPL and CS stages, the success rate is first discussed with sample outputs from both software and hardware implementations, and the reflection of software testing on hardware results. Afterwards, performance estimates for both software and hardware implementations are presented. Comparison of results with the related work is then discussed.

The algorithm proposed for the NPL stage was first implemented and tested using MATLAB, as a proof of concept, and two image databases. The two databases include HD images for cars with Qatari NPs taken under different weather and light conditions. The first database consists of 111 HD images, which was collected by the authors using digital cameras, includes both frontal and rear view images. The database used for early stage testing and development. The second database, which consists of 958 frontal-view HD images was given by the security services at Qatar University and taken from different HD cameras installed at the University gates. This database was used for final testing, as it includes images from a real-life deployed security system. Each of the security system cameras, depending on its distance from the passing cars, take images that have far view or near view of the car. The images also include different illumination conditions which are sunny view, under-shadow view, and night view.

Table VIII shows NPL testing results, which are discussed in terms of different sets in order to test the reliability of the stage under different view conditions. Fig. 11a, 11b, and 11c illustrate the conditions in these sets with samples taken from the second database, showing car images of daylight far, shadow near and night views respectively. The NPs in Fig. 11 have been hidden for privacy reasons.

Results show that NPs can be identified and localized under different illumination conditions and for cars at different distances from the camera. Test images include cars with wide NPs that have a standard size of $520\ mm \times 111\ mm$. This can be extended to different plate sizes by applying different geometrical selection conditions in the CCA sub-stage.

TABLE VIII. DETAILED HD NPL RESULTS [20]

Set	Set View	No. of Images	Success Rate
1	Daylight, far view	101	96.0
2	Daylight, near view	522	99.4
3	Shadow, near view	274	96.4
4	Night view images	61	96.7
Total		958	98.0

(a) (b)

(c)

Fig. 11. Sample images from different testing sets (a) daylight far view (b) shadow near view (c) night view

The 945 NP outputs of the HD NPL stage were used for testing the developed CS stage. The size range of the NPs was approximately from 15×80 pixels to about 45×200 pixels, depending on the camera view.

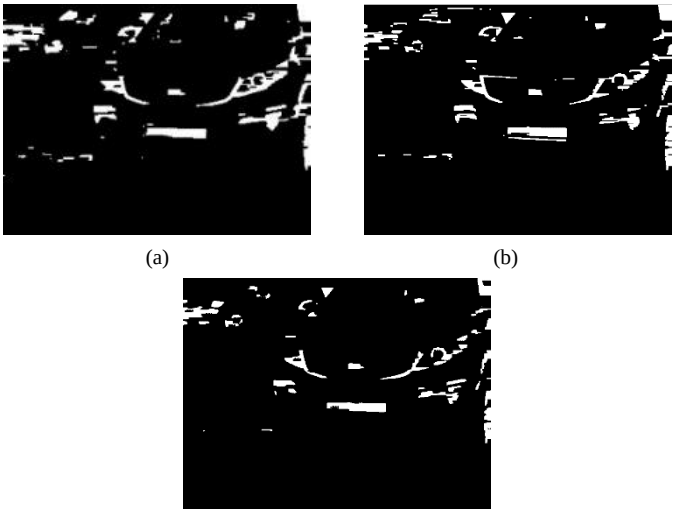
Results have shown that this algorithm has a success rate of 99.5% per plate (with Qatari NPs having 3-6 number characters per plate). Its software implementation is capable of segmenting an NP that is cropped from an HD image in 16.4 ms. The "per plate" successful segmentation rate shows how many NPs were segmented correctly. This rate reflects on the performance of this stage as a part of the full ANPR system.

Results show that this algorithm can segment irrespectively of the illumination conditions of the NP, whether under sunlight, shadow, or from night view cameras. Furthermore, due to the way it is tailored, this algorithm can also segment characters in rotated NP. This is important as no NP adjustment or rotation stages are needed before this CS stage. Detailed CS results for each set are shown in Table IX. Results are shown per-plate, i.e. how many NPs were fully segmented correctly. A mistake in the segmentation of one character deems the whole plate as a failed sample. This shows the contribution of the CS stage as a part of the entire system.

TABLE IX. DETAILED CS RESULTS

Set	Set View	No. of Plates	Success Rate (%)
1	Daylight, far view	99	98.99
2	Daylight, near view	516	98.84
3	Shadow, near view	269	99.63
4	Night view	61	98.36
Total		945	99.05

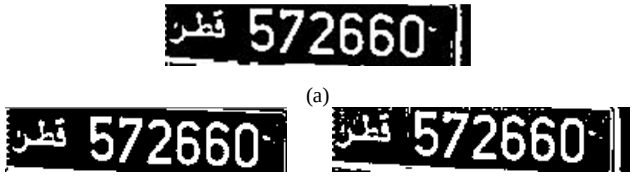
To reflect on and understand the effect of moving from software to hardware implementation on the recognition rate, sample outputs of the MATLAB, OpenCV, and the implemented PL block of the HD NPL morphological operations sub-stage are shown in Fig. 12a, 12b, and 12c respectively. Sample outputs from the MATLAB, OpenCV, and the implemented PL block of the CS stage are also shown in Fig. 13a, 13b, and 13c respectively.



(a) (b)

(c)

Fig. 12. HD NPL morphological operations sub-stage sample outputs (a) MATLAB (b) OpenCV (c) PL



(a)

(b) (c)

Fig. 13. CS Morphological Operations Sample Outputs (a) Matlab, $\Delta t = 10, w = 9$ (b) OpenCV, $\Delta t = 10, w = 9$ (c) PL, $\Delta t = 15, w = 8$

TABLE X. COMPARISON OF NPL RESULTS

NPL	Implemented Hardware	[10]	[8]	[5]	Implemented Software	[30]	[31]
Time (ms)	16.17	4.7	9.25	141.62	32.84	400	111
Success Rate (%)	98	97.8	87	96	98	97.9	96.5
Platform	Zynq-7000	FPGA Virtex-4	FPGA Virtex-2	DSP C6411 & FPGA	Corei7, 3.1 GHz	Pentium IV, 1.6 GHz	-
No. of Images	958	1307	-	100	958	1088	1344
Resolution	HD, 960×720	SD, 640×480	256×256	352×288	HD, 960×720	SD, 768×512	NA

From these samples and another good amount of randomly selected samples, it can be seen that outputs of all used platforms are quite similar. Furthermore, as the used CCA algorithm is exactly the same in both software and hardware, its output should be the same. This gives the estimation that the hardware recognition rate is equal to that of the software. A worst-case scenario would be needing some very simple fine tuning of the binarization threshold in the morphology block of HD NPL, or adjusting the CCA geometrical selection conditions a little bit on the hardware side, and this does not affect the hardware recognition rate.

Most related work focuses only on either NPL or CS algorithms. Therefore, for the sake of thorough comparison, results will be discussed and compared separately.

Table X shows a comparison between the implemented HD NPL system and other NPL systems. The total time this stage takes to process one HD image is equal to the addition of processing times on both the PL and the PS, which is equal to:

$$\begin{aligned}\text{Total Time} &= \text{Time on PS} + \text{Time on PL} \\ &= 8.09 + 8.08 = 16.17 \text{ ms}\end{aligned} \quad (2)$$

Total processing time may look slightly higher in comparison with other FPGA-only systems. This is because CCA implementation on the PL is faster than the PS. However, using the PS comes with other extra advantages such as being able to easily tune and modify the geometrical selection conditions, or add different selection parameters, e.g. contour, centroid, etc. for further improvements for this system, or when porting the basic principles and algorithms of this system to other image processing applications such as object detection and classification. Advantages also include leaving more space for implementation of other system components on the PL, and having a much faster system development time without sacrificing the real-time constrain of processing a full frame in under 40 ms. Direct comparison with the work in [10] is difficult due to the way they reported their performance. However, they report an NP localization rate of 97%. The time to run their complete ANPR system on a machine with Xeon x5650 processor and a GTX570 GPU is reported as 2 seconds.

Same way of calculating execution time applies for the CS stage, which equals to:

$$\begin{aligned}\text{Total Time} &= \text{Time on PS} + \text{Time on PL} \\ &= 0.08 + 0.51 = 0.59 \text{ ms}\end{aligned} \quad (3)$$

This is about 61% of the total processing time of the stage that is running solely on the PS.

Comparison of success rate and time performance results with other software-based and hardware-based systems is shown in Table XI. This system exceeded all other systems in terms of both speed and segmentation rate. Higher segmentation rate is owed to having the NPs cropped from HD images, and thus having a higher resolution for the NP with higher probability of having more empty space around the character with the character itself covering more pixels. This helps solve the downfalls that other systems faced like small spaces separating the characters, and not having enough character pixels in the NP image when the character is damaged, which increases the probability of error. Overcoming these downfalls is what motivated this research that covers HD ANPR systems. The nature of Qatari NPs also helped achieve such high performance for this simple CS algorithm, as all potential characters in an NP are connected characters. It is worth noting that the "resolution" column in the table refers to the resolution of the images that the NPs were extracted from, or in other words, the resolution of the images of that fully implemented ANPR system.

In all, segmented NP characters can be extracted from a car image in $16.17 + 0.59 = 16.76\ ms$. This is performed with a success rate of:

$$\text{Success Rate} = 98\% \times 98.74\% = 96.76\% \quad (4)$$

Overall hardware utilization results for the two implemented stages of the system are shown in Table XII.

TABLE XII. TOTAL HARDWARE UTILIZATION

Name	BRAM	DSP48E	FF	LUT
Total	34	19	8262	11001
Available	280	220	106400	53200
Utilization (%)	12.14	8.64	7.77	20.68

TABLE XI. COMPARISON OF CS RESULTS

CS	Implemented Hardware	[19]	[14]	[12]	Implemented Software	[12]	[19]	[16]
Time (ms)	0.59	1.8	N/A	1.4-2	16.4	23	N/A	25
Success Rate (%)	98.74	N/A	87.16	97.7	98.74	96.2	95.6	96.58
Platform	Zynq SoC	TI C64	FPGA Virtex-4	FPGA Virtex-4	Core i7, 3.1 GHz	Dual Core, 2.4 GHz	Pentium IV, 1.6 GHz	Core i5, 2.5 GHz
Resolution	HD	352 x 288	N/A	SD	HD	SD	SD, 768 x 512	300x400 - 1200x2134

This work was implemented on a Zedboard, a development board that contains the chip Zynq-7020 from the Zynq-7000 SoC family. Zynq-7020 PL includes 6650 Configurable Logic Blocks (CLBs). Each CLB includes eight LUTs and 16 FFs. In all, total CLB utilization for the two stages is 1376, or approximately 21%. This leaves plenty of area to any remaining stages or sub-stages that will be implemented on the PL for the OCR stage, or for other deployment related issues of the ANPR system. Hardware utilization of this NPL stage is somehow similar to that of [10]. It consumes 85% less than [8] for LUTs and FFs, and 75% BRAMS. This is mainly an outcome of heavily using the available DSP units in the implemented design, and to having some parts of the system running on the PS. DSP usage in [8] and [10] is not reported. Comparison of hardware utilization of the CS stage can be performed with the previous work in [12]. The design in [12] occupied 2100 slices, 3011 LUTs, 2 BRAMs and 1 DSP48s, where each CLB in Virtex-4 FPGA contains four slices, eight LUTs and eight FFs. Proposed design uses much less FFs, slightly less LUTs, no DSP48Es, but much more BRAM. BRAM usage was determined by Vivado as what the routing strategy believed was fastest. This can be controlled using optimization flags (Array_Map to reduce consumption, Array_Partition to increase processing speed). As BRAM resources are still plenty, no reduction in their count was applied. Reduction in resource utilization in this comparison can mainly be attributed to performing CCA on the PS.

Power consumption statistics are shown in Table XIII. They were estimated using Vivado design tools. Static power consumption is caused by transistor leakage current, and depends primarily on the chip manufacturing technology. Dynamic power consumption is primarily caused by transistor switching actions. It depends on the switching frequency and the chip area utilization (thus design complexity). The PS has the highest contribution to the power consumption, given its complexity and high running frequency. PL power consumption is minimal. The contribution of each stage to the overall power consumption can be approximated by observing percent utilization of each stage, shown in Tables III and VI. The HD NPL PL block consumes 4 times more power than that of the CS PL block. Previous work on NPL in [9] and on CS in [12] contains power usage statistics of the implemented stages on FPGA. However, comparison with their results is not possible due to the distinct nature of the used platforms.

TABLE XIII. ESTIMATION OF POWER CONSUMPTION

Type	Name	Power (mW)	Contribution to its Type	Overall Contribution
Dynamic	Clocks	29	2%	91%
	Signals	8	1%	
	Logic	7	1%	
	BRAM	2	<1%	
	DSP	4	<1%	
	PS	1537	94%	
Static	Device Static	158	-	9%
Total		1587		

VII. CONCLUSION

In this paper, algorithms for the NPL and CS stages of an HD ANPR system have been developed, implemented on hardware, and tested. The importance of this work lies in presenting algorithms that are designed specially to process HD car images, in addition to being fast and efficient for hardware implementation. MATLAB implementation for both algorithms has been used as a proof of concept. Zynq-7000 heterogeneous SoC device has been used for hardware implementation. Two databases, which include Qatari NPs taken under different weather and light conditions, have been used for testing. Obtained results show that both proposed algorithms outperform existing work in terms of recognition rate and implementation flexibility. The use of HD images improved the performance of the CS stage, without having an impact on the overall processing speed.

Hardware utilization of both stages is considerably low, despite them being optimized primarily for speed. This leaves a lot of free fabric area available for any upcoming system components. Achieving such good performance and utilization using Vivado HLS shows the increase in quality of auto-synthesized RTL implementations, quickly approaching those manually-written using a Hardware Description Language.

This is part of an ongoing research that aims to implement the entire HD ANPR system on an SoC, where parallelism will be exploited in the hardware implementation of the proposed algorithms to meet the real-time requirements of the system.

ACKNOWLEDGMENT

This publication was made possible by UREP grant #17-138-2-037 from the Qatar national research fund (a member of Qatar foundation). The statements made herein are solely the responsibility of the authors.

The authors would also like to thank security services at Qatar University for providing the data used in this paper.

REFERENCES

[1] Z. Jeffrey and S. Ramalingam, "High definition Licence Plate Detection Algorithm," Southeastcon, 2012 Proc. of IEEE, FL, 2012, pp. 1-6, DOI: 10.1109/SECon.2012.6196912.

[2] X. Zhai and F. Bensaali, "Standard Definition ANPR System on FPGA and an Approach to Extend it to HD," 7th IEEE GCC Conference and Exhibition, pp. 214-219, Qatar, Nov 2013, DOI: 10.1109/IEEEGCC.2013.6705778.

[3] S. Du, M. Ibrahim, M. Shehata and W. Badawy, "Automatic License Plate Recognition (ALPR): A State-of-the-Art Review," IEEE Trans. Circuits Syst. Video Technol., vol. 23, no. 2, pp. 311-325, 2013, DOI: 10.1109/TCSVT.2012.2203741.

[4] X. Zhai, PhD Dissertation: Automatic Number Plate Recognition on FPGA, University of Hertfordshire, United Kingdom, 2013.

[5] C. Arth, C. Leistner and H. Bischof, "TRIcam: an embedded platform for remote traffic surveillance," in Proc. of IEEE Computer Vision and Pattern Recognition Conference, 2006, pp. 125-134, DOI: 10.1109/CVPRW.2006.208.

[6] Xilinx. Zynq-7000 All Programable SoC. Available: http://www.xilinx.com/products/silicondevices/soc/zynq-7000.html (Accessed on April, 2016).

[7] C.Arth, F. Limberger, and H. Bischof, "Real-Time License Plate Recognition on an Embedded DSP-Platform," Presented at the IEEE

Conference on Computer Vision and Pattern Recognition, pp. 1-8, 2007, DOI: 10.1109/CVPR.2007.383412.

[8] T. Kanamori, H. Amano, M. Arai, D. Konno, T. Nanba, and Y. Ajioka, “Implementation and Evaluation of a High Speed License Plate Recognition System on an FPGA,” in 7th International Conference on Computer and Information Technology, 2007, pp. 567-572, DOI: 10.1109/CIT.2007.167.

[9] X. Zhai, F. Bensaali and S. Ramalingam, "Improved number plate localisation algorithm and its efficient field programmable gate arrays implementation," IET Circuits, Dev. & Sys., vol. 7, no. 2, pp. 93-103, 2013, DOI: 10.1049/iet-cds.2012.0064.

[10] O. Bulan, V. Kozitsky, P. Ramesh and M. Shreve, "Segmentation- and Annotation-Free License Plate Recognition With Deep Localization and Failure Identification," in IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 9, pp. 2351-2363, Sept. 2017, DOI: 10.1109/TITS.2016.2639020

[11] . Anagnostopoulos, I. Anagnostopoulos, I. Psoroulas, V. Loumos and E. Kayafas, "License Plate Recognition From Still Images and Video Sequences: A Survey", IEEE Transactions on Intelligent Transportation Systems, vol. 9, no. 3, pp. 377-391, 2008, DOI: 10.1109/TITS.2008.922938.

[12] X. Zhai and F. Bensaali, “Improved Number Plate Character Segmentation Algorithm and Its Efficient FPGA Implementation,” Journal of Real-time Image Processing, Springer, vol.10, issue 1, pp. 91-103, Mar, 2015, DOI: 10.1007/s11554-012-0258-5.

[13] M. I. Schlesinger and V. Hlavác, "Ten Lectures on Statistical and Structural Pattern Recognition," ISBN: 978-1-4020-0642-5, Springer, 2002, .

[14] H. Caner, H.S. Gecin and A.Z. Alkar, “Efficient embedded neural network based license plate recognition,” IEEE Transactions on Vehicular Technology vol. 57, pp. 2675-2683, 2008, DOI: 10.1109/TVT.2008.915524.

[15] Y. Youngwoo, B. Kyu-Dae, Y. Hosub, and K. Jaehong, “Blob extraction based character segmentation method for automatic license plate recognition system,” in IEEE International Conference on Systems, Man, and Cybernetics, 2011, pp. 2192-2196, DOI: 10.1109/ICSMC.2011.6084002.

[16] Y. Liu, H. Huang, J. Cao and T. Huang, “Convolutional neural networks-based intelligent recognition of Chinese license plates,” Journal of Soft Computing, Springer, pp. 1-17, FEB, 2017, DOI: 10.1007/s00500-017-2503-0.

[17] K.-B. Kim, S.-W. Jang and C.-K. Kim, “Recognition of Car License Plate by Using Dynamical Thresholding Method and Enhanced Neural Networkss,” in Computer Analysis of Images and Patterns. vol. 2756, N. Petkov and M. Westenberg, Eds., ed: Springer Berlin/Heidelberg, 2003, DOI: 10.1007/978-3-540-45179-2_39.

[18] Y. Cui and Q. Huang, “Extracting characters of license plates from video sequences,” Machine Vision and Applications, vol. 10, no. 5-6, pp. 308–320, 1998, DOI: 10.1007/s001380050081.

[19] C. N. E. Anagnostopoulos, I. E. Anagnostopoulos, V. Loumos and E. Kayafas, "A License Plate Recognition Algorithm for Intelligent Transportation System Applications," IEEE Transactions on Intelligent Transportation Systems, vol. 7, pp. 377-392, 2006, DOI: 10.1109/TITS.2006.880641.

[20] O. Hommos, A. Al-Qahtani, A. Farhat, A. Al-Zawqari, F. Bensaali, A. Amira and X. Zhai, “HD Qatari ANPR system,” CIICS16, Dubai, United Arab Emirates, 2016, pp. 1-5, DOI: 10.1109/ICCSII.2016.7462420.

[21] R. Gonzalez and R. Woods, Digital image processing. New Delhi: Dorling Kindersley, 2014. ISBN: 0-13-168728-x 978-0-13-168728-8.

[22] A. Ismail and M. Hassaballah, “Image Feature Detectors and Descriptors: Foundations and Applications,” Studies in Computational Intelligence Series, Springer, 2016, DOI: 10.1007/978-3-319-28854-3.

[23] Zedboard.org. Community-based site. Available: http://zedboard.org/ (Accessed on December, 2016).

[24] Xilinx. Accelerating OpenCV Aplications with Zynq-7000 All Programable SoC using Vivado HLS Video Libraries. Available: http://www.xilinx.com/support/documentation/application_notes/xapp1167.pdf (Accessed on April, 2016).

[25] Linaro. Official site. Available: http://www.linaro.org/ (Accessed on December, 2016).

[26] Xilinx. AXI Video Direct Memory Access v6.2. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdm a.pdf (Accessed on Apr. 2016).

[27] Xilinx. Vivado High-Level Synthesis. Available: http://www.xilinx.com/products/designtools/vivado/integration/esl-design.html (Accessed on April, 2016).

[28] Xilinx. HLS Video Library. Available: http://www.wiki.xilinx.com/HLS+Video+Library (Accessed on Apr. 2016).

[29] Xilinx. Vivado Design Suite User Guide High-Level Synthesis Available:https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf (Accessed on Sept. 2017).

[30] S.L. Chang, L.S. Chen, Y.C. Chung and S.W. Chen, "Automatic License Plate Recognition," IEEE Transactions on Intelligent Transportation Systems, vol. 5, pp. 42-53, 2004, DOI: 10.1109/TITS.2004.825086.

[31] A. Clemens, L. Florian, B. Horst, “Real-time license plate recognition on an embedded DSP-platform,” Proc. IEEE Computer Vision and Pattern Recognition Conf., pp. 1–8. (2007), DOI: 10.1109/CVPR.2007.383412.