International Journal of Parallel Programming (2021) 49:253–284
https://doi.org/10.1007/s10766-020-00690-y

# *i*DocChip: A Configurable Hardware Architecture for Historical Document Image Processing

## Multiresolution Morphology-based Text and Image Segmentation

**Menbere Kina Tekleyohannes**[1] · **Vladimir Rybalkin**[1] · **Muhammad Mohsin Ghaffar**[1] · **Javier Alejandro Varela**[1] · **Norbert Wehn**[1] · **Andreas Dengel**[2]

Received: 28 October 2019 / Accepted: 7 December 2020 / Published online: 30 January 2021

**Abstract**
In recent years, optical character recognition (OCR) systems have been used to digitally preserve historical archives. To transcribe historical archives into a machine-readable form, first, the documents are scanned, then an OCR is applied. In order to digitize documents without the need to remove them from where they are archived, it is valuable to have a portable device that combines scanning and OCR capabilities. Nowadays, there exist many commercial and open-source document digitization techniques, which are optimized for contemporary documents. However, they fail to give sufficient text recognition accuracy for transcribing historical documents due to the severe quality degradation of such documents. On the contrary, the anyOCR system, which is designed to mainly digitize historical documents, provides high accuracy. However, this comes at a cost of high computational complexity resulting in long runtime and high power consumption. To tackle these challenges, we propose a low power energy-efficient accelerator with real-time capabilities called iDocChip, which is a configurable hybrid hardware-software programmable System-on-Chip (SoC) based on anyOCR for digitizing historical documents. In this paper, we focus on one of the most crucial processing steps in the anyOCR system: *Text and Image Segmentation*, which makes use of a multi-resolution morphology-based algorithm. Moreover, an optimized FPGA-based hybrid architecture of this anyOCR step along with its optimized software implementations are presented. We demonstrate our results on multiple embedded and general-purpose platforms with respect to runtime and power consumption. The resulting hardware accelerator outperforms the existing anyOCR by ***6.2*** $\times$, while achieving ***207*** $\times$ higher energy-efficiency and maintaining its high accuracy.

Extended author information available on the last page of the article

**Keywords** Optical character recognition · Text and image segmentation · FPGA · Hardware-software co-design · Zynq · Hardware architecture

## 1 Introduction

National Archives, libraries, and museums around the world contain numerous collections of historical documents. Nowadays, due to the increasing demand to electronically access and preserve these valuable resources, the research area to digitize document contents into machine-readable text has been amplified. Transcription of documents also leads to the development of further applications such as text mining, document indexing, word spotting, text-to-speech conversion, etc. To digitize the historical archives, the documents are first scanned then an OCR is applied on each document image. Many libraries are equipped with special devices that are specifically tailored for scanning historical documents. However, these machines are usually large and stationary. Challenges arise when instant word spotting, document indexing, etc are required. As a result, there is a high demand for a portable hand-held OCR device that can transcribe historical documents in real-time with high accuracy. The first requirement in the design of such a device is selecting an algorithm that gives high accuracy for digitizing historical archives. The portable device also needs to have low power consumption and high energy efficiency. Moreover, in order to achieve real-time constraints, the design must provide high performance and high-throughput, see Sect. 4.

Nowadays, there exist several commercial and open-source OCR engines, such as ABBYY [1], Omnipage [2], OCRopus [3], Tesseract [4], etc., which are optimized to transcribe contemporary documents like modern books, letters, memos, and other documents. However, unlike contemporary documents, historical archives are subject to quality degradation caused due to non-uniform shading, bleed-through, complex irregular layouts, skewed/overlapping texts and even physically damaged/missing portions of pages. As a result, the readily available OCR engines fall short of giving acceptable accuracy for historical document images. In order to overcome the quality degradation and achieve high text recognition accuracy for historical archives, Bukhari et al. [5] introduced anyOCR. This end-to-end OCR system is designed to digitize both contemporary and historical documents. It mainly emphasizes transcribing historical archives with high accuracy by including techniques that are able to overcome high-quality degradations. The anyOCR system is an open-source software that consists of three image pre-processing and layout analysis steps (*Binarization*, *Text and Image Segmentation*, *Text-line Extraction*) and machine learning-based *Text-line Recognition*. The overall pipeline is shown in Fig. 1.

Given a highly degraded historical Latin document images dataset [6], anyOCR achieves a 75.92% accuracy while the commercially available and open-source OCR systems ABBYY and Tesseract achieve only a 66.47% and 56.83% accuracy, respectively. However, the higher accuracy of anyOCR comes at the cost of high computational complexity. Each of the three preprocessing steps, in anyOCR,

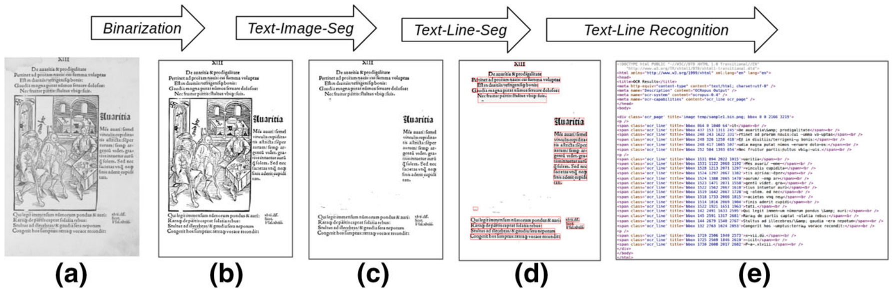

**Fig. 1** The anyOCR system processing pipeline, including Fig. 1a original gray-scale input image, Fig. 1b after *Binarization*, Fig. 1c after *Text and Image Segmentation*, Fig. 1d after *Text-line Extraction,* and Fig. 1e after *Text-line Recognition* steps

consists of several computer vision and image processing algorithms such as morphological and other filtering operations, connected component analysis, feature extractions, and many other mathematical operations. Moreover, the character recognition step is based on a deep recurrent neural network. All these complex image processing techniques result in high computational complexity causing long runtime and high power consumption when the anyOCR system runs on conventional computing platforms like CPUs.

In anyOCR, different operations require different precision. For example, for an 8-bit grayscale input image to the anyOCR pipeline, the internal operations may be binary, 2-bit, 3-bit or other arbitrary precision computations. However, CPU and GPU support only a limited number of data types, such as int8, int16, etc. On the contrary, custom hardware platforms like FPGAs and ASICs support arbitrary precision computations. Hence, the flexibility provided by these platforms allows for the development of architectures that can benefit from operations on single bits that cannot otherwise be handled efficiently on general-purpose computing platforms. Since the traditional software-centric computing platforms fail to meet our requirements, we target a custom hardware architecture, which can be implemented on an FPGA or ASIC. Such systems provide customized hardware acceleration for critical tasks, custom memory hierarchy, and other features that can be exploited to increase the overall efficiency. However, ASICs are reasonable only for high volume production due to high manufacturing costs and design efforts.

The anyOCR system is parameterizable for each new dataset. Hence, in order to support a variety of document recognition applications with different configurations, the OCR engine has to adapt to changes even after the production of the device, which makes FPGAs compelling since they support reconfiguration. FPGAs also enable the development of high-performance and high energy-efficient systems, hence fulfilling all the requirements for designing the portable OCR device. In order to provide further flexibility, in the past few years, FPGA vendors have introduced an SoC which integrates the software programmability of a processor with the hardware configurability of an FPGA. Therefore, we target the Zynq-7000 All Programmable SoC, specifically the Xilinx Zynq-7000 XC7Z045 device, to design the anyOCR-based document image processing system that we called the iDocChip. Through efficient hardware-software partitioning, the capability of the programmable fabric and the embedded CPU is exploited.

In this paper, we focus on the second step of anyOCR pipeline, *Text and Image Segmentation*, which separates a document page into text and non-text components. This step involves several filters and computer vision algorithms like connected component labeling and feature extraction that require high external memory bandwidth to transfer data between computational units and memory. The input images and the operations performed in this pipeline stage have binary pixels. However, there is still high bandwidth overhead due to the high resolution of the images. For example, our dataset has image dimensions of 2166x3219 pixels with 300 dpi. To achieve high throughput, the data transfers between the computational units on the FPGA fabric and the off-chip memory should be minimized. This implies that intermediate results must be buffered on the on-chip memory. However, due to the limited number of available memory units, storing all of the intermediate results is not feasible, even for the largest available FPGA fabric. Hence, after analyzing the different parallelization schemes and studying the trade-offs between different types of memories on an FPGA, the best fitting hardware-software partitioning scheme is chosen. Accordingly, the first hybrid hardware-software architecture of the multiresolution morphology-based *Text and Image Segmentation* technique is developed [7] that is part of iDocChip. In this paper, we present the optimized FPGA-based hybrid architecture of this anyOCR step and its optimized software implementations.

The rest of the paper is organized as follows. In Sect. 2, related works are discussed, followed by details of our previous works in Sect. 3. Then the constraints of iDocChip and the novel contributions of this paper are stated in Sect. 4. After a brief discussion of the text-image segmentation algorithm in Sect. 5, the hardware architecture is detailed in Sect. 6. Then evaluation and results are presented in Sect. 7. Finally, Sect. 8 gives an outlook and concludes the paper.

## 2 Related Works

In this chapter, we review related works from 4 different perspectives. First, works on cross-platform comparisons that make use of image processing algorithms are presented. Second, an overview of end-to-end OCR pipelines is given. Finally, different algorithms and hardware architectures for *Text and image segmentation* are reviewed.

### 2.1 Cross-Platform Comparison

To design the portable OCR device, different computing platforms were considered. The design space included embedded CPUs, embedded GPUs, FPGAs, and ASICs. By targeting image processing algorithms, many publications have presented comprehensive comparisons of different processing platforms [8–12].

Brugger et al. [8] has made a cross-platform analysis of morphological operations implemented on low power platforms: a GPU (NVIDIA Tegra K1 SoC), CPU (Low-Power Intel Core i7-4790T), and FPGA (Xilinx Zynq 7020), and compared their performance and energy efficiency. The authors observed that the filtering algorithms implemented on the GPU are 5$\times$ slower than a similar implementation on the CPU, while the FPGA offers high throughput, comparable to the CPUs. Moreover, the authors showed that the FPGA implementation of the morphological operations is 8-10$\times$ more energy-efficient compared to the CPU and GPU implementations.

In a recent study, Qasaimeh et al. [9] conducted a benchmark of run-time performance and energy-efficiency for different vision algorithms implemented on three commonly used hardware accelerators for embedded vision applications: ARM57 CPU, Jetson TX2 GPU, and ZCU102 FPGA. For optimal performance, the authors used the vendor optimized vision libraries: OpenCV, VisionWorks, and xfOpenCV, respectively. Their results show that while kernels that are simple and easy-to-parallelize perform well on GPUs with 1.1-3.2$\times$ energy/frame reduction compared to CPU and FPGA, for complete vision pipelines, FPGA outperforms the others with energy/frame reduction of 1.2–22.3$\times$. Moreover, authors also observed that as the complexity of the vision pipeline grows, the FPGA performs increasingly better than the CPU and GPU.

In [10], Page and Mohsenin have presented a pulse wave spectral Doppler ultrasound imaging system implemented on a Virtex-5 FPGA and in 65nm CMOS ASIC design for performance comparisons. The ASIC implementation consumes 27$\times$ less power and can run at a much higher clock frequency, 3$\times$ higher than the FPGA implementation. The Virtex-5 design requires 1159 of 17, 280 slice resources and consumes 1.089$W$ of power when running at its maximum clock speed of 333$MHz$. The ASIC design has an area of 0.573$mm^2$ and consumes 41$mW$ of power at a maximum clock speed of 1$GHz$. On the other hand, FPGAs provide an economically suitable reconfigurable platform whereas the cost of producing the ASIC design would only be feasible for large-scale production. The authors concluded that the FPGA design had comparable efficiency and performance compared to the ASIC implementation while having the advantage of being cost-effective and reconfigurable.

Due to their software-hardware programmability, for our work in iDocChip, we target the Zynq-7000 All Programmable SoC. These devices are equipped with dual-core ARM Cortex$^{TM}$-A9 processors integrated with 28nm Artix-7 or Kintex-7 based programmable logic (PL). The processor cores, also known as processing system (PS), allow customization, and therefore together with the PL, they provide a flexible SoC on a single chip. These devices enable highly differentiated designs for a wide range of embedded applications, including medical endoscopes, professional cameras, machine vision, and many others [13–16]. In particular, the Xilinx Zynq-7000 XC7Z045 SoC, which contains Kintex-7 based PL alongside the ARM cores, is being used for diverse portable industrial applications [17–19].

## 2.2 Complete OCR Processing Pipeline

In the literature, there exist several works regarding modern contemporary and historical document processing. Many of these works involve deep learning approaches that are usually preceded and followed by pre- and post-processing [20–24]. Pre-processing is performed on the input image, and it enhances the image, making it suitable for character classification. Various operations are involved in this process, such as skew detection and correction, noise detection and removal, binarization, thinning, and normalization. Then page segmentation is performed to identify patterns of non-text regions, text lines, words, and characters from the document. These segmented areas are then extracted using feature extraction. The classifier module labels characters through supervised learning. Finally, post-processing is performed to correct possible errors. Nowadays, the pre- and post-processing steps are also being processed by using neural networks [25–27]. Although these methodologies are very promising, due to the large amount of parameters, they are very challenging to translate into hardware-aware implementations without loss of accuracy.

### 2.3 Text and Image Segmentation Algorithms

Over the years, different techniques have been proposed targeting *Text and Image Segmentation*. In literature, this task is also referred to as text-localization, text and non-text separation, text detection and extraction, or suppression of non-text components. A comprehensive survey of *Text and Image Segmentation* algorithms is given in [28–30]. Methods used for this task are generally classified into four groups: region-based, pixel-based, component-based, or multiresolution morphology-based methods.

Region-based segmentation is the most widely reported method of classification, where page segmentation is performed to extract homogeneous regions before classifying the text and non-text zones. Oyedotun and Khashman [31] presented a top-down approach, where segmentation goes from a coarse level to a finer level, such that large homogeneous regions are first extracted, and then refinement is performed at the subsequent levels. Authors segmented the document into regions by shifting a mask over a grayscale input image. From each segmented region, first- and second-order statistical features are extracted. The regions are then classified by using feed-forward neural networks.

In a pixel-based *Text and Image Segmentation* approach, however, classification techniques are applied on each pixel of a document to form a region. Hence this type of segmentation is called bottom-up. In [32], Moll et al. have classified individual pixels in order to avoid restrictiveness of region shapes. The authors then investigated different classification techniques, including brute-force k-Nearest Neighbors (kNN), fast approximate kNN using hashed k-d tress, classification and regression trees, and locality-sensitive hashing.

However, region and pixel-based classification methods are often very sensitive to initial region segmentation result. Therefore, they are susceptible to an erroneous text and non-text region classification. In the case of a document with a text inside graphics/images, these methods may result in regions containing pictures with embedded text. Hence, such approaches may fail as they make the classification decision for the entire region.

The third classification method uses connected component analysis (CCA) to separate text and image segments from the input document image. Bukhari et al. [33] perform segmentation by finding and labeling connected components (CCs) and then extracting a set of features from them. Finally, a multi-layer perceptron (MLP) classifier is used to segment the components as either text or non-text. Here, the authors have assumed that the context of the text component is always structured However, this is not the case every time since a text may appear next to an image that is out of context and random. Hence, misclassifications may occur.

Additional to the above classification approaches, a hybrid method exists which combines both region and component-based approaches for *Text and Image Segmentation*. In [34], Chowdhury et al. presented a method where texture analysis is performed to separate halftones of a grayscale image. Then after binarization of the resulting image, CCs are extracted. Pixel density information is computed from the CCs, and it is used to classify the text characters. However, these methods are computationally exhaustive and not very efficient.

The last method is a multiresolution morphology-based classification, which is introduced by Bloomberg [35]. The author applies a series of morphological operations to sub-sampled images in order to progressively remove text components and preserve portions of halftone images. Then the text region is retrieved by masking the halftone images. Although this method performs well for halftone mask segmentation, it fails to accurately segment other non-text elements such as drawings, maps, etc. Hence, Bukhari et al. [36] presented an improved multiresolution morphology-based text and non-text segmentation algorithm by first reconstructing the broken drawing lines and applying the hole-filling morphological operation. As a result, the algorithm can accurately separate text and non-text images, including halftones, drawings, logos, graphs, maps, etc.

### 2.4 Text and Image Segmentation Hardware Architectures

Many works, such as those presented above, concentrate more on the methods of *Text and Image Segmentation*, and they lack insight in the performance of the approach. Hence, many of the existing algorithms suffer from a flawed tradeoff between accuracy and speed. There have only been few proposals for accelerating this task on dedicated hardware. Kumar et al. [37] used a method based on a discrete wavelet transform (DWT) to detect and extract text from a document image. The authors designed the architecture and implemented the system on a Virtex-5 FPGA. For a dataset of 33 images, the authors were able to achieve 96 seconds. However, the energy efficiency or power consumption of the system is not openly communicated.

Bai et al. [38] proposed a novel architecture of a convolutional neural network called MSP-Net for text/non-text image classification. The MSP-Net consists of 4 main parts: image-level feature generation, multi-scale spatial partition, block-level representation generation, and text/non-text block classification sub-network. The system takes the whole image as an input and outputs block-level classification results in an end-to-end manner. An NVIDIA GTX TitanX GPU was used for training purposes.

The research area for hardware acceleration of text and image segmentation has not been well investigated, which makes quantitative comparison difficult. However, in our previous work [7], we have presented, the first hardware architecture for multiresolution morphology-based text and image segmentation (see Sect. 3). Here, we extend our previous work through optimizations and further contributions (see Sect. 4).

## 3 Our Previous Works

As shown in Fig. 1, the anyOCR pipeline includes layout analysis and character recognition steps. In our previous works [7, 39–41], hardware accelerators of these steps were introduced and implemented on Xilinx Zynq-7000 XC7Z045 SoC. The performance and energy-efficiency of each one of these accelerators were compared to the original Python-based anyOCR running on Intel® Core™ i7. The hardware architectures of these algorithms serve as part of iDocChip. As mentioned above, iDocChip [5] is the FPGA-based end-to-end OCR system based on anyOCR. Brief explanations of these four hardware accelerators are given below.

### 3.1 Binarization

The first pipeline step of anyOCR converts the input grayscale image into binary. For this, percentile-based binarization (PBB) is used. This method is suitable to binarize document images with non-uniform illuminations. The heterogeneous hardware-software architecture of this step is given in [39]. Here, the PBB algorithm is partitioned into hardware and software parts for an efficient hardware-software co-design. With this implementation, both PL and PS of Zynq run concurrently and communicate through General Purpose (GP) or High Performance (HP) interfaces. The PL uses 64-bit bi-directional HP AXI ports to communicate with the PS. Custom direct memory access (DMA) controllers are implemented to transfer data between PS and PL. Moreover, based on the binarization architecture, a hybrid hardware-software FPGA-based accelerator was presented. When running at 166*MHz*, this accelerator outperforms the original anyOCR software implementation by a factor of 20 in terms of runtime performance while achieving an energy-efficiency of 10 *Images*/*J*, which is much higher than what the low power embedded processors (ARM Cortex-A9, ARM Cortex-A53) achieved.

### 3.2 Text and Image Segmentation

This pipeline step separates the text and non-text parts of a document image, using a multiresolution morphology-based text and image segmentation method. The corresponding hybrid hardware-software architecture was presented in [7], which optimizes the original software algorithm of anyOCR in order to make it suitable for efficient hardware parallelization with negligible loss of accuracy. To exploit the hardware concurrency, time-critical operations with parallelization capability are implemented in the hardware fabric, while those operations that are inherently sequential are implemented in software, resulting in an efficient hardware-software partitioning of the text-image segmentation algorithm. Then, by using this architecture, a hybrid FPGA-based accelerator was implemented on Zynq XC7Z045 SoC. The system-level architecture is similar to Fig. 16. However, the software part of the hybrid architecture is very inefficient due to the inherently sequential union-find algorithm that is used for the connected component labeling (CCL) part of the *Text and Image Segmentation* chain. The presented accelerator in [7] has achieved nearly a 3× gain in performance and more than 125× increase in energy-efficiency compared to the original anyOCR implementation. In the current paper, this previous work is further optimized, and better performance and energy-efficiency are achieved (see Sect. 7).

## 3.3 Text Line Extraction

The final preprocessing step extracts the text lines using a Gaussian smoothing-based algorithm. This method contains four major processing blocks, mainly to estimate text scale, separate columns, find text lines, and extract the lines from the image. Since this step involves several filters that contribute to 85% of the runtime of the text line extraction pipeline step, the original software algorithm of anyOCR has been highly optimized in order to design an efficient hardware architecture, with a negligible loss of accuracy. Then, this heterogeneous architecture is used to implement a hybrid accelerator for the text line extraction step on a Zynq XC7Z045 SoC. The robust accelerator outperforms the original anyOCR software implementation by 135× in terms of runtime performance and over 1100× in energy efficiency.

## 3.4 Text Line Recognition

The anyOCR system uses Bidirectional LSTM (BiLSTM) Neural Network with Connectionist Temporal Classification (CTC) for the text-line recognition. In [41], the first hardware architecture of the BiLSTM network with CTC was presented for

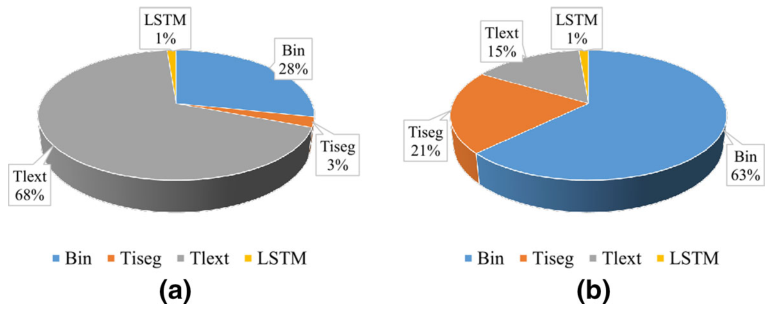

**Fig. 2** Performance comparison of processing steps measured in terms of runtime in *ms*. Fig. 2a shows the runtime of anyOCR pipeline steps when the system is running on Intel Core i7. Fig. 2b shows the runtime of iDocChip pipeline steps implemented on Zynq 7045 at 166 *MHz*. The four pipeline steps are denoted as *Bin* for Binarization, *Tiseg* for Text and Image Segmentation, *Tlext* for Text line Extraction, and *LSTM* for LSTM-based Text line Recognition

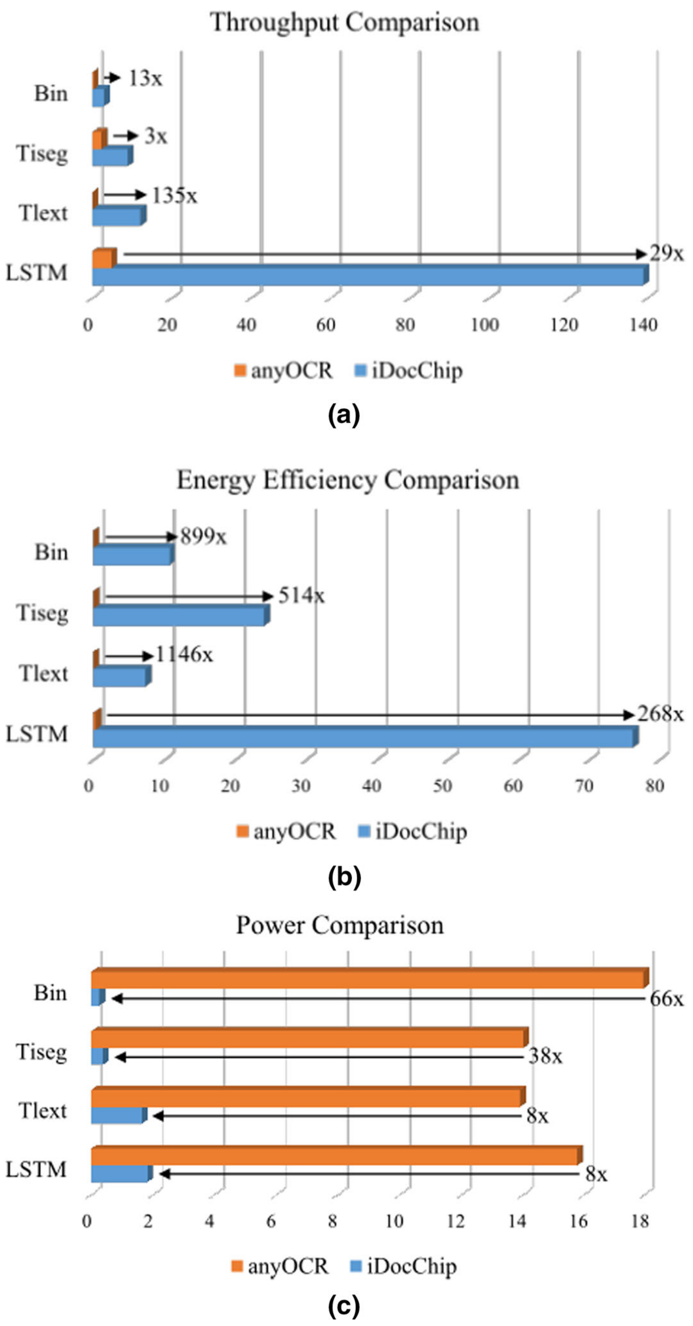

**Fig. 3** Comparison of anyOCR and iDocChip processing pipelines in terms of Fig. 3a throughput (*FPS*), Fig. 3b energy efficiency (*FPS*/*Watt*), Fig. 3c power consumption (*Watts*). *Bin* stands for Binarization, *Tiseg* for Text and Image Segmentation, *Tlext* for Text line Extraction, and *LSTM* for LSTM-based Text line Recognition

OCR. The paper took into account two application scenarios on-line and off-line OCR. In the first case, real-time processing is required where images appear one at a time. Thus, the system has been optimized to process a single image. In the second case, batch processing is performed where several images are processed in parallel with separate instances of the accelerator. The text line recognition architecture is then implemented on Xilinx Zynq XC7Z045 SoC. It achieves a 459× higher throughput than state-of-the-art implementation [42], and it can process up to 6 images in parallel.

As shown in Fig. 2b, the *Text and Image Segmentation* step is the second most time-consuming task, next to binarization contributing to 21% of the overall processing time of iDocChip. The software pipeline of this step takes the second least amount of time as shown in Fig. 2a. Compared to the anyOCR process, the *Text and Image Segmentation* accelerator gives a significantly higher energy efficiency and throughput values while consuming a smaller amount of power, see Fig. 3.

## 4 Constraints and Novel Contributions

In order to design the portable hand-held OCR device that meets the thermal design power (TDP) requirements, four main design constraints are set. These include power (in *watts*), energy-efficiency (in *frames per second* (*FPS*) *per watt*), performance (in terms of runtime (execution time) in *ms*), and throughput (in *FPS*). For our real-time document image processing system, the hardware design is expected to process the complete OCR pipeline in 500*ms* for a single image. This timing constraint enables instant word spotting and other OCR applications. As a result, a minimum of 2*FPS* throughput is expected. Moreover, the power budget of the device is limited to 2*W*. Hence, energy-efficiency is expected to be at least 1*FPS*/*W*. Moreover, to make sure that portability is achieved, the design must fit within the available resources of the portable Xilinx Zynq XC7Z045 SoC device. The constraints are summarized in Table 1.

Compared to the original anyOCR, the already existing iDocChip implementations of the OCR pipelines, overall, have gained 30× in performance. More significantly, it has achieved 467× energy efficiency. Although the system is able to

**Table 1** Constraints and results from our previous work for the complete historical document analysis pipeline

| | Performance Runtime (ms) | Power (W) | Throughput (FPS) | Energy Efficiency (FPS/W) |
|---|---|---|---|---|
| Original anyOCR | 16471.8 | 15.42 | 0.09 | 0.006 |
| Constraints | $<500$ | $<2$ | $>2$ | $>1$ |
| Existing iDocChip | 544.84 | 1.03 | 2.9 | 2.8 |

fit into a portable device due to its compact design and low power consumption, it fails by almost 9× to give the required performance. Since the *Text and Image Segmentation* pipeline step is the second time-costly process, it is further optimized to meet the design requirements.

Therefore, in this work, we focus on the hybrid hardware-software implementation of multiresolution morphology-based *Text and Image Segmentation* method. To summarize, the novel contributions of this paper are:

- A faster software implementation of the *Text and Image Segmentation* of anyOCR algorithm is presented.
- The first hybrid accelerator for *Multiresolution Morphology-based Text and Image Segmentation* [7] is further optimized and is tested on the historical Latin document images dataset [6].
- By optimizing the software part of the first hybrid accelerator [7], 40% of runtime is reduced, and 46% of energy efficiency is increased. Hence, the performance requirements of the hand-held device are met.
- The new optimized hardware accelerator outperforms the original anyOCR software implementation (running on i7-4702MQ) in terms of runtime and energy efficiency by 6× and 207×, respectively.
- The performance and energy efficiency of the *Multiresolution Morphology-based Text and Image Segmentation* algorithm is compared across different platforms, including low-power CPUs.

## 5 *Text and Image Segmentation* Algorithm

The *Text and Image Segmentation* algorithm is based on multiresolution morphology, which involves basic and compound morphological operations. For high-resolution images, these operations require large structuring elements that are computationally intensive. Hence, multi-scale image representations are used for efficient analysis of image contents as well as for speeding up the image processing operations. The multi-scale transform is employed through threshold reduction.

The output image of the percentile-based binarization (PBB) algorithm [39], which performs binarization of the anyOCR input image, is used as the input for the *Text and Image Segmentation* pipeline. As shown in Fig. 4, this binary input image, where the foreground and background pixels are represented by ’0’ and ’1’, respectively, is inverted. After inversion, the image is processed by two threshold reduction operations with thresholds equal to one. This operation subsamples the input image, while preserving the density of low as well as high frequency components within the document image. The subsample of the inverted image is computed by replacing each 2x2 block of four pixels by a single pixel, depending on the chosen threshold ($T$). The value of $T$ can be between one and four. The resulting pixel value of the threshold operation is ’1’ if the sum of the values of the four pixels in the block is greater than or equal to $T$. Otherwise the resulting pixel value is set to ’0’.

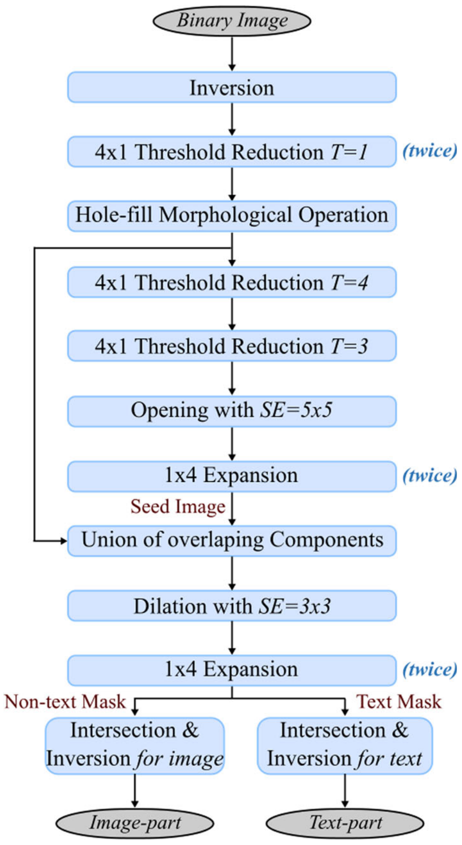
Binary Image
Inversion
4x1 Threshold Reduction T=1
(twice)
Hole-fill Morphological Operation
4x1 Threshold Reduction T=4
4x1 Threshold Reduction T=3
Opening with SE=5x5
1x4 Expansion
(twice)
Seed Image
Union of overlaping Components
Dilation with SE=3x3
1x4 Expansion
(twice)
Non-text Mask
Text Mask
Intersection & Inversion for image
Intersection & Inversion for text
Image-part
Text-part

**Fig. 4** The *Text and Image Segmentation* algorithm

The subsampled image is then processed with a hole-filling morphological operation to fill hollow contours that are usually presented in drawings, maps, and graphs. The resulting image of the hole-fill operation is further resized using two threshold reduction operations with *T* equal to four and three. Next, a morphological opening using a 5x5 structuring element is performed. These steps remove the text components and preserve some portions of the halftone components. The resulting image is referred to as the seed image. Following this, two expansion operations are applied to resize the seed image to the subsampled image from the hole-filling step. This algorithm uses a trivial expansion method in which each pixel value is copied into a 2x2 pixel block of four pixels.

In the next process step, the halftone mask image is generated by comparing the hole-fill result with the seed image. This requires connected component labeling (CCL) to be performed on the output image of the hole-fill operation. After CCL, only those components that fully or partially overlap to the seed image are selected. Next, morphological dilation with a structuring element of 3x3 is applied.

Finally, the halftone mask image is further expanded by two expansion operations to obtain the dimensions of the original image. The resulting image contains the non-text mask for the inverted input binary image. The mask for the text-part is obtained by inverting the non-text mask image of the segmentation. The final outputs are gathered by intersecting each of these text-part and image-part masks with the inverted input binary image and complimenting the results.

## 6 Hardware Architecture

The first hybrid hardware-software architecture of multiresolution morphology-based *Text and Image Segmentation* is presented in [7]. Efficient partitioning of the algorithm into hardware and software is essential for an effective hardware-software co-design. The time-critical operations with high parallelization capability are implemented in the hardware fabric to exploit the hardware concurrency. While inherently sequential operations are implemented in software. Figure 5 shows the best (in terms of resources/performance) hardware-software partitioning we found

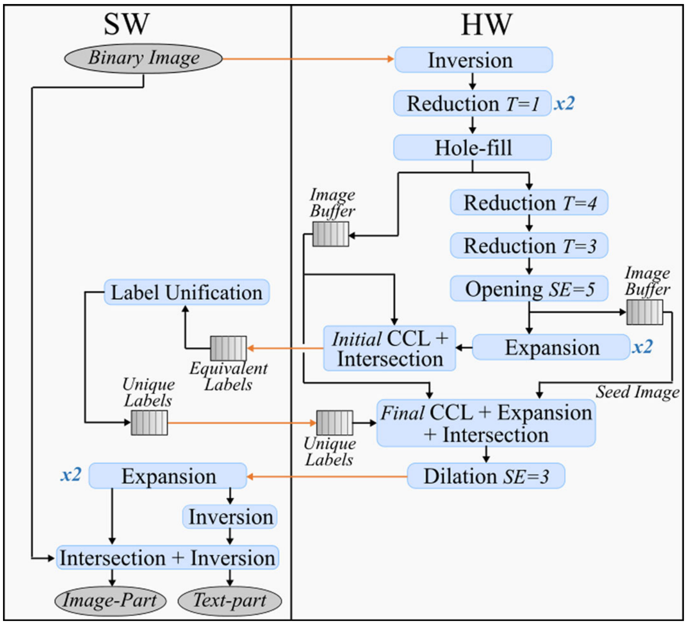

**Fig. 5** Hardware/software partitioning of *Text and Image Segmentation* algorithm based on multiresolution morphology

for this hybrid architecture. The hardware architectures of each of the operations are explained in detail below.

## 6.1 Inversion

The first operation of the *Text and Image Segmentation* algorithm is inversion. This task, which inverts the input binary image, is processed in hardware. Hence, taking into account an interface with a 64-bit data width, the CPU core sends a set of 64 pixels in each transaction into the hardware. Since the cost of logic blocks is minimal, by instantiating 64 inverters (NOT logic gates), a set of 64 pixels are processed in a single clock cycle. Therefore, the throughput of this step is 64 pixels per clock cycle (PPC).

## 6.2 Reduction with Threshold $T = 1$

Reduction operation computes on blocks of 2x2 pixels that spread over two rows of an image. When the pixels are streamed into the fabric in a raster-scan manner, they are required to be buffered on the on-chip memory in order to produce output results. Although the required buffer size is rather small due to binary pixels, this overhead is unnecessary. Moreover, with pixels streaming in a raster

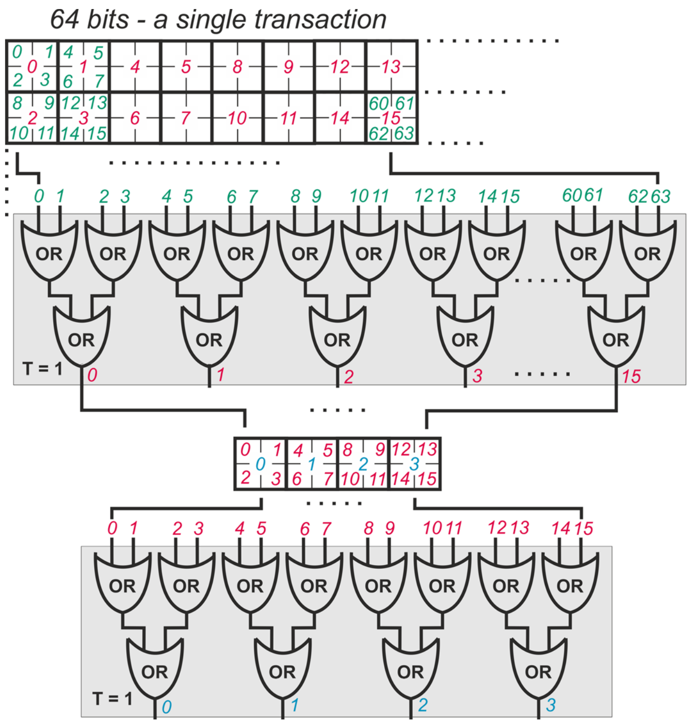

**Fig. 6** The modified memory access pattern with 2 threshold reduction operations of *T=1*

scan manner, the first and second reduction operations with *T=1* will generate results after reading every second and fourth row of the binary input image, respectively. This makes the datapath unbalanced from a workload point of view and reduces the throughput. To alleviate this issue, different memory access and store patterns are used. Hence, the binary input image (the output of PBB) is stored in the offline memory in blocks instead of row by row. In each clock cycle, taking into account the 64-bit wide interface, 64 pixels are transferred in a 2x2 block over 4 rows and 16 columns. The mapping is depicted in Fig. 6. Since the first operation (inversion) computes on pixel level, its results are not affected by any access patterns. As a result of the modified memory access pattern, the threshold reduction operations deliver a regular throughput. This approach also eliminates the requirement of on-chip intermediate buffers. Furthermore, the next processing function, i.e., the hole-fill operation, receives pixels streaming in a raster-scan manner. The throughputs of the first and second reduction operations with *T=1* are 16 and 4 pixels per clock cycle, respectively. The dimension of the resulting image is reduced to a quarter of the binary input image.

## 6.3 Hole-Filling

Hole-filling makes use of mathematical morphological operations: erosion and dilation. Erosion removes pixels from the object boundaries of the given image, while dilation adds pixels. The anyOCR system uses morphological reconstruction by erosion to fill the holes within the image. This operation is an iterative process where a large number of raster-scans are required. For example, for the dataset images given in [6], these scans range from 198 to 827 for a 4-connected structuring element (window). However, such a process with a large number of iterations is not feasible to design in hardware. Hence, to overcome this issue, an alternative custom algorithm is used, which achieves the same results as morphological reconstruction by erosion in fewer iterations. Moreover, unlike this highly iterative process, the alternative hole-fill algorithm achieves sufficient quality in each sequence-run. Hence, the algorithm is able to exit even before the final iteration without largely affecting the overall output image of the Text and Image Segmentation algorithm.

In an alternate hole-fill algorithm, each image sequence-run can be of two types: 4-direction sequence or 2-direction sequence, as shown in Fig. 7. To process a 4 sequence-run (Fig. 7a) for a given image of size H*W, the algorithm is computed by scanning the image in 4 directions. First in direction 1: left to right, top to bottom

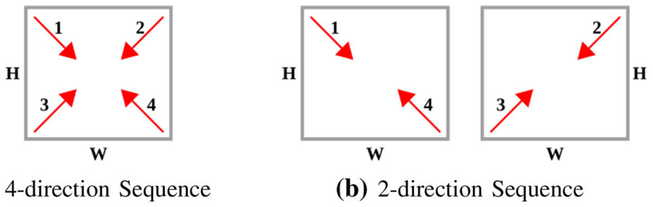

**Fig. 7** The two types of image scan directions for an alternate hole-fill algorithm

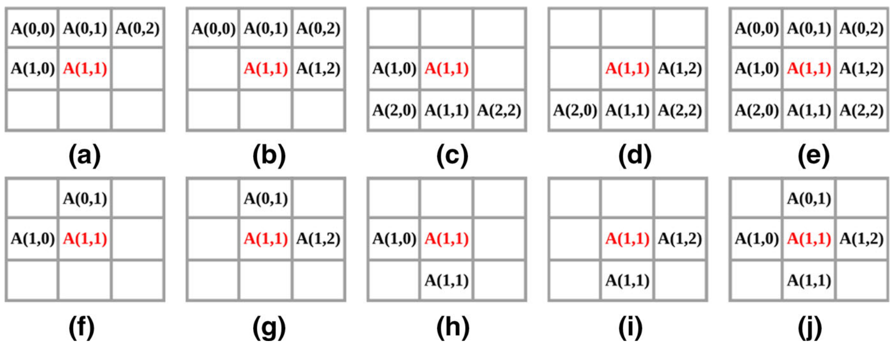

**Fig. 8** Computation windows for different directions using a 4/8 full connectivity (Fig. 8a–e) and a 2/4 cross connectivity (Fig. 8f–j). Fig. 8a–d show windows for 4-fully-connective computations for processing in direction-1,2,3 and 4 respectively. Fig. 8e shows the window for an 8-fully-connective computation. While, Fig. 8f–i show windows for 2-connective computations for processing in direction-1,2,3 and 4 respectively. Figure 8j shows the window for 4-cross-connective computation, which can be used in any direction

(raster-scan), then in direction 2: right to left, top to bottom, then in direction 3: left to right, bottom to top and finally in direction 4: right to left, bottom to top (anti-raster scan). However, for the 2-direction sequence-run (Fig. 7b), the algorithm is only computed in two opposing directions (directions 1 and 4 or directions 2 and 3). Due to the sequential nature of the algorithm, the sequence of directions impacts the number of iterations and convergence of the result.

Similar to the morphological reconstruction approach, the alternative hole-fill algorithm creates a mask image to start the hole-filling process. The mask has the dimension of the input image. Its border pixels are set to zero, while all internal pixels are set to one. The input image is used together with the initial mask to compute the algorithm. Furthermore, similar to flood-fill, which determines the area connected to a given node in a multi-dimensional array, the alternative hole-fill algorithm makes connectivity-based processing. It supports 2-, 4- or 8-connective computations. These computation windows are shown in Fig. 8.

The input image *I* of the alternative hole-fill algorithm is the output of the second reduction step. The mask image *M* is created on the fly. If the pixel at the image *I(i,j)* is 0, the algorithm reads the adjacent window values from the mask image. If any of the pixels in the window are 0, then the current pixel of the mask, *M(i,j)*, is replaced by 0. In such a manner, the algorithm runs through the complete image. After completely updating the mask image for the given direction run (sequence-run), the algorithm then starts to process the follow-up direction, according to the chosen image-scan sequence.

The alternative hole-fill algorithm offers a trade-off between latency of the operation and quality of the result. This is possible through tunning the number of iterations, alternating the image scan directions, or changing the shape of processing window (2-/4-/8-connective window). With a number of iterations, the output of the alternative hole-fill algorithm converges to the output of the original reconstruction by erosion. However, to achieve the same result, the number of

**Table 2** Comparison of morphological reconstruction and the proposed alternative hole-fill algorithm, in terms of the required number of image scans (best / worst cases) and hardware resources, considering the dataset images given in [6]

| | Num. iter. | | Speedup | Hardware resources | | | |
|---|---|---|---|---|---|---|---|
| | Worst | Best | | LUT | FF | BRAM | DSP |
| *Recon. by Eros.* | | | | | | | |
| 4-conn. | 827 | 198 | 1 | 2713 | 1811 | 36 | 1 |
| 8-conn. | 699 | 182 | 1 | 2795 | 1925 | 36 | 1 |
| *Proposed, 4-dir.* | | | | | | | |
| 4-conn. | 21 | 5 | 39x | 1092 | 876 | 41 | 3 |
| 8-conn. | 22 | 4 | 31x–45x | 1321 | 1043 | 41 | 4 |
| *Proposed, 2-dir.* | | | | | | | |
| 4-conn. | 65 | 5 | 12x–39x | 956 | 765 | 41 | 3 |
| 8-conn. | 49 | 4 | 14x–45x | 1016 | 817 | 41 | 3 |

Num. iter. refers to the number of image scans the algorithms took to achieve the same result

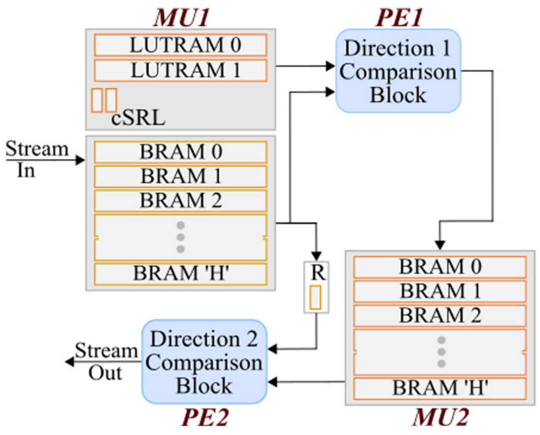

**Fig. 9** Hardware architecture of the alternative fill-hole algorithm. MU1 and MU2 are the memory units, and PE1 and PE2 are the processing engines

iterations required for the alternative hole-fill algorithm is much less compared to that of morphological reconstruction. This analysis is shown in Table 2.

For our *Text and Image Segmentation* algorithm in iDocChip, we chose the alternate hole-fill algorithm to process in two directions, i.e., raster scan followed by anti-raster scan, using a 4-cross-connected window. The hardware architecture of this algorithm for the 2-direction sequence is shown in Fig. 9. As shown in Table 2, the resource consumption of the implemented alternate hole-fill architecture for the 4-cross-connective algorithm is less than all other computation types, including the original hole-fill algorithm (morphological reconstruction by erosion). Moreover,

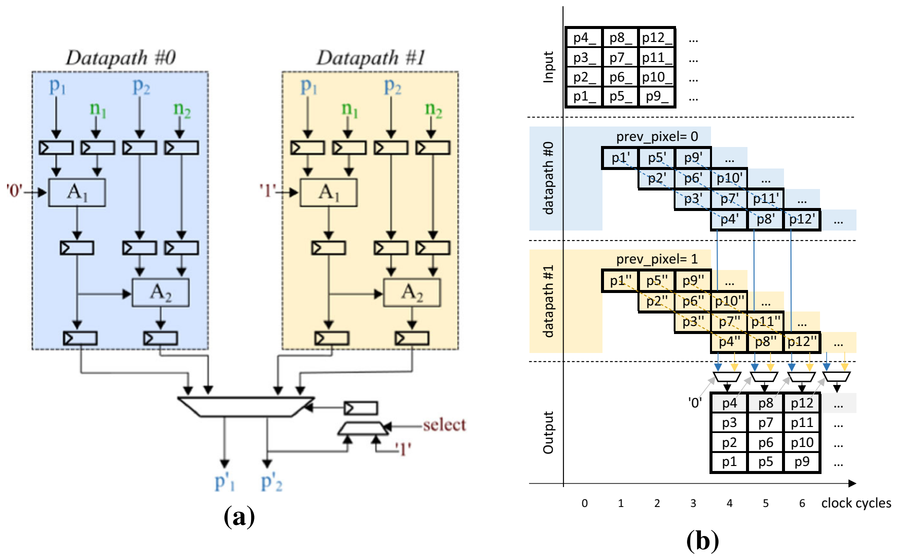

**Fig. 10** Architectural optimizations for alternative hole-fill algorithm. Figure 10a shows data paths with parameterizable width. Here, $(p_1, p_2)$ are input pixels read together, $(n_1, n_2)$ are the corresponding neighbors, and $(p'_1, p'_2)$ are the corresponding output (processed) pixels. "Select" is used to input '1' for every set of pixels that start exactly at the boundary of the image. Fig. 10b shows the temporal behavior of two 4-pixel-wide data paths

this alternative hole-fill algorithm gives an approximate output that results in a <1% degradation of OCR recognition accuracy while requiring only two image-scans.

In this stream-based system, pixels are received from the second reduction stage of the *Text and Image Segmentation* pipeline in a raster-scan manner with 4 pixels per clock cycle. However, the alternative hole-fill algorithm is sequential in nature; hence, the result of the previous pixel affects the results of the following pixels. To exploit hardware parallelization, we propose two architectural optimizations. These optimizations aim at maximizing the throughput and minimizing the overall image-level initiation interval of the alternative hole-fill algorithm.

The first optimization allows us to process multiple pixels in parallel by defining data paths with parameterizable width. Figure 10a presents the solution for two pixels *(p1,p2)* processed in parallel, and it can be scaled to any number of pixels (in our implementation to 4). Since the alternative hole-fill algorithm is sequential, the result from the previous pixel (here p0) affects the result of p1, while the output of p1 is required to compute p2, and so on. To alleviate this issue, Fig. 10a shows a pipelined design based on the carry select adder where two parallel data paths compute the output for the current pixel under different assumptions of the result of the previous pixel. As shown in Fig. 10a, Datapath #0 assumes the result of the previous pixel to be '0', and Datapath #1 assumes it to be '1'. When the result of the previous pixel is ready, the correct output is selected through the multiplexer. This way, we can process a new set of pixels every consecutive clock cycle. To illustrate this, Fig. 10b presents the temporal behavior of these two data paths for a width of 4 pixels.

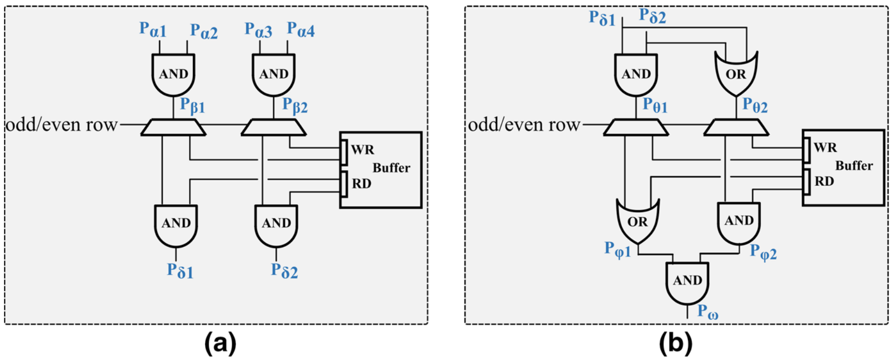

**Fig. 11** Architecture of the threshold reductions with $T$=4 in Fig. 11a and $T$=3 in Fig. 11b

The second optimization is possible when the number of image scans is more than one. In this case, we can place enough ping-pong buffers for the working images so that the input images can be processed one after another without delay.

Using these algorithmic optimizations, the hole-fill operand processes a new set of pixels every clock cycle. Therefore, through this design, an output throughput of 4 pixels per clock cycle is achieved for the hole-fill algorithm. The resulting image is stored internally for further processing.

## 6.4 Reduction with Threshold $T = 4$ and $T = 3$

As discussed in Sect. 6.2, the threshold reduction operation computes on adjacent pixels: first column-wise and then row-wise. For the reduction with $T=4$, the input pixels are streamed in a raster-scan manner. Hence, a line buffer is used to store column-wise results before performing the row-wise operation. For threshold reduction, the raster-scan processing scheme results in a lower output throughput. The reduction operation with $T=4$ receives 4 pixels in each clock cycle from the hole-fill algorithm and outputs 2 pixels per clock cycle for every even row of the incoming image, see Fig. 11. Similarly, the reduction with $T=3$ takes these 2 pixels in each clock cycle and outputs 1 pixel per clock cycle for every fourth row, compared to the image coming from the hole-fill operation. As shown in Fig. 11, the threshold reduction architecture with $T=4$ uses *AND* logic blocks. The comparison units are modified accordingly for reduction with $T=3$.

## 6.5 Morphological Operations (Opening and Dilation)

In this stage, highly-parallelized atomic morphological operations (erosion and dilation) are implemented based on [43]. In this *Text and Image Segmentation* pipeline, opening and dilation use 8-connective windows with 5x5 and 3x3 structuring elements, respectively. Morphological operations have kernel-decomposition property that allows erosion/dilation computations to be processed using two consecutive 1-dimension windows, namely horizontal and vertical windows. For example, a 5x5 opening operation is decomposed into a 1x5 window followed

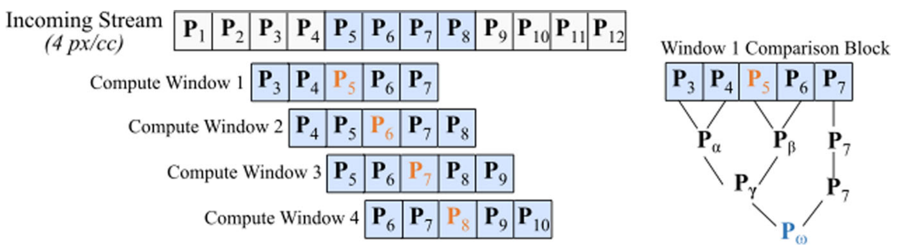

**Fig. 12** Computation of a morphological operation with a 5x5 structuring element. This computation assumes an input image streaming at 4 pixels per clock cycle, which involves 4 compute windows. The comparison block of window 1 (on the right-hand side) shows the tree-structure process of the morphological operation

by a 5x1 window computations. Moreover, similar to the hole-fill algorithm, it supports high-throughput computations for image streams with more than one incoming pixel. Figure 12 shows the main comparison block of morphological operation with a structuring element (window) size of 5, assuming the input image streaming at 4 pixels per clock cycle. By processing the 4 compute windows in parallel using different comparison blocks, the morphological operation architecture achieves 4 pixels per clock cycles.

However, for the implemented *Text and Image Segmentation* pipeline, reduction with threshold $T=3$ stage only results in 1 pixel per clock cycle for every fourth row of the hole-fill operation result. Consequently, the opening morphological process is implemented with a similar throughput. For future computations, the throughput must be balanced in order to obtain a regular output in every clock cycle. As a result the output image of the opening morphological operation is stored internally using on-chip buffers.

### 6.6 Expansion

This step expands the result of the morphological opening operation. This step copies one pixel into a block of four pixels. Since it uses an already buffered input image, the expansion operation achieves a regular 1 pixel per clock cycle (PPC).

### 6.7 Connected Component Labeling (CCL) and Seed Intersection

This step of the *Text and Image Segmentation* algorithm is the most critical in separating the text-part of the document from the image-part. It involves two layers of computation. Moreover, in this step, two images are used as input: $I_{HF}$ (the output image from the hole-fill operation) and $I_{EX}$ (the expanded image after morphological opening, i.e. seed image). First, the connected components that exist on $I_{HF}$ are labeled. This process is called CCL, then the output of CCL is intersected to the seed image $I_{EX}$. Both CCL and seed intersection each involve two raster-scans. Since the number of provisional labels is very high, building a hardware-only solution for these operations requires extensive hardware resources. To overcome this challenge, a hybrid hardware-software solution is presented that is based on the

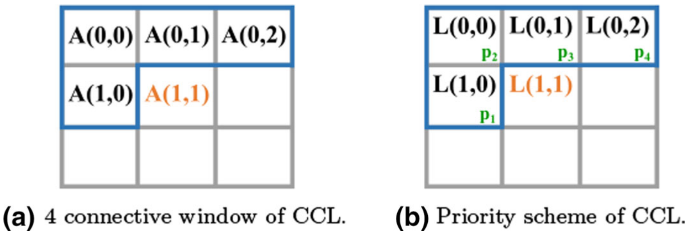

**Fig. 13** Computation window and priority scheme of connected component labeling. In Fig. 13a the blue box presents the window of pixels considered to find a label for pixel A(1,1). In Fig. 13b L(0,0)–L(1,1) present the labels for the pixels given in Fig. 13a, while $p_1$—$p_4$ show the priority scheme of the labels. If pixel A(1,1) is not zero, its label L(1,1) takes the non-zero label of the adjacent neighbor with the highest priority. Otherwise, if the label of all four adjacent neighbors is zero, then a new label is given to L(1,1)

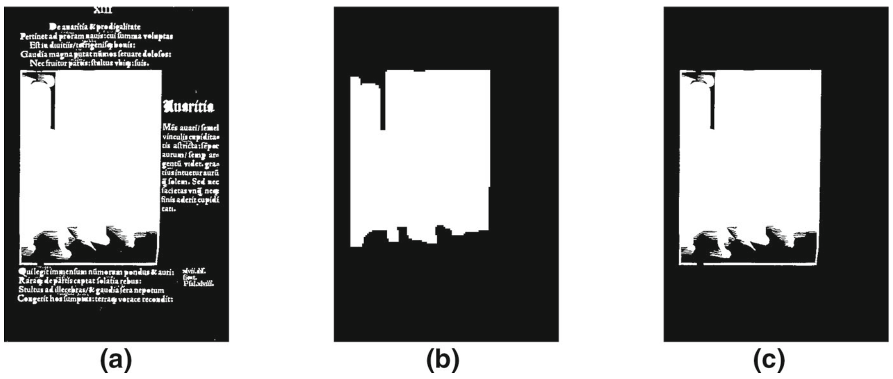

**Fig. 14** Connected component labeling and seed intersection for input image given in Fig. 1b. Figure 14a shows the output of hole-fill ($I_{HF}$) Fig. 14b shows the output of expansion (seed image/$I_{Ex}$) and Fig. 14c shows the final result of CCL and seed intersection

connected component analysis architecture given in [44]. Moreover, in order to achieve high throughput, both CCL and seed intersection are processed at the same time. Hence, the overall CCL and seed intersection operation require only two raster scans in total, while the software process of this step requires four image scans and a unification step.

CCL uses a 4-connective window, as shown in blue in Fig. 13a. In the initial scan of CCL, to find provisional labels of $I_{HF}$, each foreground pixel (non-zero pixel) of $I_{HF}$ is given a label using the priority scheme of the window, shown in Fig. 13b. Label conflicts exist when several labels represent the same connected component. Multiple representations of a component affect further processes and distort the output image of the *Text and Image Segmentation* algorithm. Therefore, to resolve this issue, any conflicting labels are recorded using an equivalence table. At the end of the first scan, equivalent labels are relabeled through a unification process, where similar labels point to the smallest label (root label) and the root label indexes ’0’. Then in the second raster-scan, each pixel is given a final label by referencing its

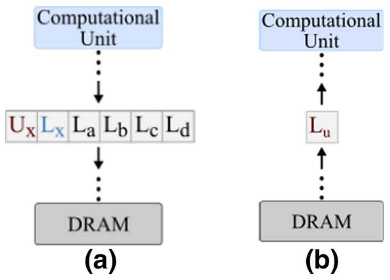

**Fig. 15** CCL data transfer from FPGA fabric to DRAM and back. Fig. 15a shows the CCL out-stream from the computational unit to the DRAM. For pixel $A_x$, the result of seed intersection (U$x$), current pixel label $L_x$, and all neighboring labels $L_a$ to $L_d$ are transferred to the DRAM through a CPU. Fig. 15b shows the CCL in-stream from the DRAM to the computational units. After CPU processes unification, all labels that are flagged unique ($L_u$) are streamed to the fabric

provisional label and the equivalence table. As an example, the CCL and seed intersection of the input image to the *Text and Image Segmentation* given in Fig. 1b is shown in Fig. 14. Here, CCL is computed for the output of hole-fill operation (Fig. 14a). After labeling each foreground pixel of this image, the total number of provisional labels is 2800. Then after applying unification, the final number of labeled connected components becomes 567. For some images, the number of provisional labels is even higher. For example, our dataset [6] has images with an original resolution of 2166$\times$3219. Due to the multi-resolution processes, however, at this stage of the *Text and Image Segmentation* algorithm, the images are resized to an 504$\times$804 resolution. To compute CCL, the maximum number of connected components that could theoretically exist for such an image with an 504$\times$804 resolution is 108741. Each label requires a 17-bit. Since it is not feasible to store a 2D array of this size (108741$\times$108741 of 17-bit values) within the hardware, the equivalent labels are stored externally.

In addition to the provisional labels, the initial seed intersection is processed. Here, the connected components that overlap with the seed image are computed by intersecting $I_{HF}$ with $I_{Ex}$. If there is an overlap, the connected component label of $I_{HF}$ at the overlapping pixel position is flagged using the unique flag $U$. Then provisional labels together with the unique flag are streamed to the external memory (DRAM) through the CPU, as shown in Fig. 15a. Unification of equivalent labels is a sequential and iterative task. If processed in hardware, it has no gain in terms of performance, and it is very costly of resources. Hence, equivalent labels are merged through a software-based computation.

### *CCL CCL Software Optimizations*

Unlike the original hybrid architecture [7], where the software process waits for all the labels to be streamed from the hardware to start unification, in this work, computation to merge equivalent labels begins as the first data arrives. Hence, the unification process is partially overlapped with the data transfer. Moreover, in the previous version of our hybrid accelerator, for unifying labels, a union-find algorithm was used where the *FindRootLabel*($x$) function follows a chain of parent labels from Label $L_x$ up the tree until it reaches a root label. However, in this modified version, a binary-tree based algorithm is used where the root labels are found much faster. Then, all unique labels are separately stored. The CPU sends these labels that are flagged unique to the FPGA fabric. These two optimization schemes reduce the runtime of the overall *Text and Image Segmentation* algorithm by 40% since the most time-consuming part of the hybrid hardware-software implementation is the latency caused by the algorithms running in software.

In preparation for the second raster-scan of CCL and seed intersection operations, images $I_{HF}$ and output of the morphological opening operation ($I_{MO}$) are stored internally using on-chip memory. However, the initially computed labeled mask, which contains provisional labels, is not stored internally as it takes a significant amount of resources. Buffering $I_{MO}$ instead of $I_{Ex}$ saves storage space since the size of $I_{MO}$ is one-fourth of $I_{Ex}$.

In the second raster-scan of CCL and seed intersection operations, first, the $I_{MO}$ image is populated on the fly to give $I_{Ex}$. Then provisional labels are recomputed using the buffered $I_{HF}$ in parallel to the intersection operation. The latter outputs pixel ‘1’ for each pixel in $I_{HF}$ where its label $L$ is an element of the unique labels $U$. Otherwise, it outputs the corresponding seed pixel from $I_{Ex}$. The throughput of this final CCL and seed intersection operation result is independent of previous blocks since the inputs of this stage are all buffered and can be streamed all at once. By studying the trade-off between resource utilization Vs. throughput, where the higher the number of pixel streams per clock cycle results in the larger resource consumption, we chose a four-pixel stream per clock cycle for this stage of the *Text and Image Segmentation* algorithm. Hence, the final CCL and seed intersection operation has an output throughput of 4 pixels per clock cycle.

### 6.8 Result Extraction

After the union of overlapping components, morphological dilation is performed to dilate important image objects. The implementation is explained in Sect. 6.5. This final block has an input/output throughput of 4 pixels per clock cycle. The result is then grouped into 64 bits and sent to the CPU, which directly expands the result into 4x4 blocks and stores it into DRAM. Then, the resulting non-text mask is intersected with the input binary image. Finally, the output is inverted to give the final image-part. Similarly, the text-part is extracted by first inverting the non-text mask and applying intersection with binary image and inversion.

## 7 Experimental Setup and Results

For experimental purposes, we used the historical Latin document images dataset [6] in order to compare the reference anyOCR system with our hardware and optimized software implementations. The original anyOCR system is a Python-based software, which uses a multi-dimensional image processing python library [45] and runs on a multi-threaded Intel Core i7-4702MQ with Turbo Boost up to 3.2

**Table 3** Configurations of different platforms for software-based tests

| Platform | Num. cores | Threads per core | Total threads | Freq. [GHz] | Tested on | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Python base. | Python opt. | C++ (ST) | C++ MT |
| i7 4702MQ | 4 | 2 | 8 | 2.9 | ✓ | ✓ | ✓ | ✓ |
| Atom C2758 | 8 | 1 | 8 | 2.4 | | ✓ | ✓ | ✓ |
| Cortex A53 | 4 | 1 | 4 | 1.5 | | ✓ | ✓ | ✓ |
| Cortex A9 | 2 | 1 | 2 | 0.8 | | ✓ | ✓ | ✓ |

GHz for 1 active core and 2.9 GHz for 4 active cores. For further analysis, the runtime and energy-efficiency of optimized software implementations are examined on different platforms, see Table 3.

## 7.1 Hardware Setup

To evaluate the hybrid hardware-software architecture of the *Text and Image Segmentation* algorithm, each hardware-based operation of the architecture is designed using Xilinx Vivado High-Level Synthesis tool of version 2018.1. It is implemented using the Vivado block design targeting the Zynq-7000 All Programmable SoC, specifically Zynq 7045 that features the xc7z045ffg900-2 FPGA fabric and dual-core ARM Cortex-A9 processors. The ARM CPU runs Linaro Ubuntu Linux version 16.1. The software-programmable parameters, such as kernel values of filters, are transferred from the Programmable System (PS) through a General Purpose (GP) port using an AXI-Lite interface. Moreover, a custom DMA is implemented. This DMA is responsible for reads/writes of data from/to DRAM using AXI memory-mapped non-coherent interface and converting each transaction into AXI stream. This DMA transfers streams in bursts of 64 bits. Hence the hardware cores are able to exploit datapath pipelining, where multiple operations are overlapped during execution, which increase the throughput. The software part of this hybrid architecture is implemented in C++ and runs on the ARM Cortex-A9 processor cores of the Zynq 7045 device at 800 MHz, while the FPGA fabric runs at 166 MHz. To evaluate the runtime and energy consumption of the proposed text and image segmentation system, the hybrid hardware-software implementation is tested on ZC-706 Evaluation Kit, which contains the target device. The system block diagram of the Zynq implementation of the text and image segmentation pipeline is shown in Fig. 16.

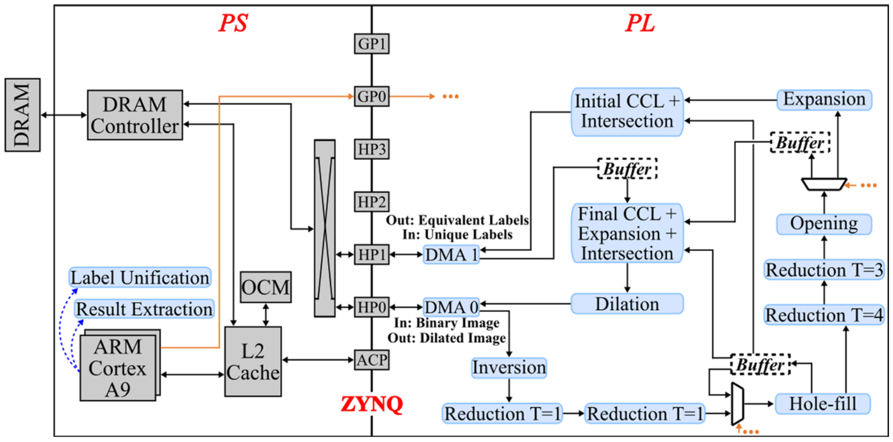

**Fig. 16** Zynq implementation of *Text and Image Segmentation* hardware architecture

## 7.2 Software Optimizations

For a fair comparison of the hardware and software implementations, different platform-dependent and algorithmic optimizations are applied on the reference anyOCR chain. These optimasations accelerate the original *Text and Image Segmentation* process. From the algorithmic side, particularly, the morphological reconstruction algorithm used for *hole-filling* operation is replaced by the alternate hole-fill algorithm. Hence, for the optimized Python implementation, the alternate hole-fill algorithm is written and compiled as a c-library, which outperforms the original implementation. Moreover, morphological operations are computed using a Manhattan-distance based algorithm for higher performance. The unification operation of CCL is also changed from a union-find based search to a binary-tree based search algorithm.

In the multi-threaded C++ implementation, image-level coarse-grained parallelization is applied using OpenMP API. In this method, all available threads process separate images at each point in time. The coarse-grained approach has appeared to be more efficient than the fine-grained parallelization applied on the level of functions or loops because it does not require any inter-core or inter-thread synchronization. Using dynamic scheduling in OpenMP, the threads start processing the next image right when they finish with the current image. This avoids idling threads. All software implementations are complied with gcc 7.4.0 and -O3 optimization flag.

## 7.3 Energy Consumption

The previously mentioned different computing platforms have different hardware setups, peripherals, and accelerators. These contribute to the total power consumption. For a fair comparison of energy among the different computing platforms, only the consumed energy $E_{comp}$ is considered. Therefore, the average active power

($P_{act}$) is calculated by subtracting the idle power ($P_{idle}$) from the total power consumption of the system ($P_{tot}$), processing all of the images in the dataset:

$$P_{act} = (P_{tot} - P_{idle}) \tag{1}$$

$$E_{comp} = P_{act} \times Runtime \tag{2}$$

The idle power is the power consumption of the complete system in the idle state. By removing this value, the influence of the power consumption of the unwanted hardware resources is minimized. The *Ecomp*, however, includes energy consumed for extra cooling caused by intense computations, which is unavoidable. Similarly for the Zynq platform, $P_{idle}$ is the power consumption of the complete board when the CPU is idle and the FPGA is not configured. The power is measured physically using a digital wall socket power meter Voltcraft VC-870.

## 7.4 Results and Discussion

Table 4 shows the runtime, power, energy, frames-per-second, and energy efficiency comparisons of the software and hardware implementations of the *Text and Image Segmentation* process on different platforms. Compared to the Python-based baseline and optimized implementations on the Core i7, the optimized hybrid hardware-software implementation of the *Text and Image Segmentation* provides a speedup of ***6.2*** $\times$ and ***4.5*** $\times$, respectively. Moreover, the optimized hardware implementation also gives a speedup of ***3.4*** $\times$, ***2.4*** $\times$, and ***1.7*** $\times$ compared to the multi-thread C++ implementation of the algorithm running on Cortex A9, Cortex A53, and the original Zynq implementation [7], respectively. Most notably, this new Zynq implementation achieves a higher energy efficiency, providing an improvement of ***207*** $\times$ and ***1.5*** $\times$ compared to the baseline anyOCR system and the original Zynq implementation [7], respectively. The graphical analysis of these results is shown in Fig. 17.

As shown in Table 4, the C++ implementations in Core i7 and Atom C2758 have higher performance compared to the Zynq implementation. The performance of the hybrid accelerator is affected by the software-part of the implementation. When the *Text and Image Segmentation* algorithm is processed on Zynq, the hardware-part takes only 15% of the total runtime of the system, while the rest 85% of the runtime is spent in the CPU. Nevertheless, the power consumptions of the Core i7 and Atom C2758 processors are much higher than the power required by the Zynq implementation. As a result, Zynq achieves the highest energy efficiency as compared to any of the software implementations, see Fig. 17.

From the accuracy point of view, the impact of *Text and Image Segmentation* on the OCR accuracy is evaluated by computing Character Error Rate (CER) compared to the ground truth while keeping all other pipeline stages similar to the original anyOCR. As shown in Table 4, the accuracy of the hardware implementation is comparable to the baseline anyOCR. Moreover, for the given dataset [39], the accuracy of the implemented *Text and Image Segmentation* substantially

**Table 4** Comparison of the *Text and Image Segmentation* process among different platforms. Table 4 shows the results of the hardware implementation of the algorithm on the Zynq 7045 device and software implementations of the algorithm on Intel Core i7 4702MQ, Intel Atom C2758, Cortex A53, and Cortex A9

| Platf. | Implementation | Runtime, [ms] | Power, [W] | Energy, [J] | FPS | Ener. eff., [FPS/W] | EDP | Acc., [%] |
|---|---|---|---|---|---|---|---|---|
| i7 | Pyt. Base. | 418.98 | 14.01 | 5.87 | 2.4 | 0.17 | 2.5 | 76.3 |
| 4702 | Pyt. Opt. | 308.24 | 17.27 | 5.32 | 3.2 | 0.19 | 1.6 | 75.71 |
| MQ | C++ (ST) | 34.06 | 12.42 | 0.42 | 29.4 | 2.4 | 0.014 | 75.68 |
| | C++ (MT*) | 11.29 | 31.89 | 0.36 | 88.6 | 2.8 | 0.004 | |
| Atom | Pyt. Opt. | 978.34 | 3.216 | 3.15 | 1 | 0.3 | 3.1 | 75.71 |
| C2758 | C++ (ST) | 91.31 | 2.24 | 0.20 | 10.9 | 4.9 | 0.018 | 75.68 |
| | C++ (MT) | 27.26 | 5.84 | 0.16 | 36.7 | 6.3 | 0.004 | |
| Cortex | Pyt. Opt. | 4504.91 | 0.57 | 2.57 | 0.22 | 0.4 | 11.6 | 75.71 |
| A53 | C++ (ST) | 481.62 | 0.32 | 0.15 | 2.08 | 6.5 | 0.072 | 75.68 |
| | C++ (MT) | 166.33 | 0.59 | 0.09 | 6 | 10.2 | 0.015 | |
| Cortex | Pyt. Opt. | 5652.55 | 0.21 | 1.19 | 0.12 | 0.6 | 6.7 | 75.71 |
| A9 | C++ (ST) | 975.4 | 0.35 | 0.34 | 1 | 2.9 | 0.332 | 75.68 |
| | C++ (MT) | 227.2 | 0.46 | 0.10 | 4.4 | 9.6 | 0.023 | |
| Zynq | FPGA Orig. | 112.94 | 0.37 | 0.04 | 8.9 | 24.1 | 0.004 | 75.70 |
| 7045 | FPGA Opt. | 67.76 | 0.42 | 0.03 | 14.8 | 35.2 | 0.002 | |

Results are shown for baseline and optimized Python, single-threaded (ST), and multi-threaded (MT) C++ implementations running on Core i7-4702MQ, Intel Atom C2758, Cortex A53, and Cortex A9. Here, runtime and energy ($E_{comp}$) are given per image, and the energy efficiency shows frames-per-second per watt (FPS/W). The OCR accuracy is given in Acc and the energy-delay product in EDP. For a fair comparison, the results for Core i7-4702MQ, MT* shows the multi-threaded implementation with simultaneous multi-threading (SMT) disabled

* Multi-threaded implementation with simultaneous multi-threading (SMT) disabled

outperforms the commercially available OCR engines ABBYY and Tesseract, which have recognition accuracy of 66.47% and 56.83%, respectively.

As shown in Fig. 1, the anyOCR system has four distinct parts, binarization, *Text and Image Segmentation*, text-line extraction, and text/character recognition. The total resource consumption of the four OCR pipelines implemented on Xilinx Zynq 7045 is given in Table 5. The results show that almost half of the resources on the portable FPGA-based SoC are free. Therefore, the unused resources can be utilized to couple the four different blocks together in order to build a streaming end-to-end OCR system. This suggests that the overall design is able to fit in the embedded portable device (Xilinx Zynq 7045) without any issue, fulfilling the design requirement of portability.

When revisiting the design specifications, the hardware implementation achieves the goal for an energy-efficient portable device with a $1FPS/W$ and $2FPS$, under the constrained power of $2W$. Moreover, unlike the previous Zynq implementation [7], this optimized design achieves the $500ms$ runtime (performance) constraint by

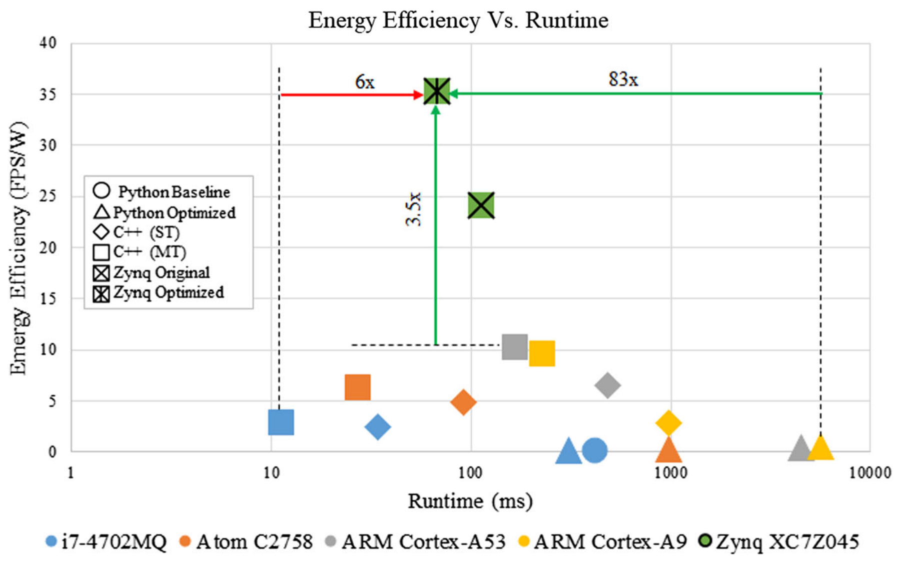

**Fig. 17** Comparison of Energy Efficiency Vs. Runtime on different platforms. Gains and losses are noted by green and red arrows, respectively

**Table 5** Resource utilization of the hardware implementation for binarization [39], text-image segmentation (this work) , text line extraction [40], and character recognition [41], on the Zynq 7045 device @ 166MHz

| Pipeline | LUT | FF | BRAM 36Kb | DSP |
|---|---|---|---|---|
| Binarization | 8358 (4%) | 11039 (3%) | 45 (9%) | 1 (1%) |
| Text-image segmen. | 51009 (23%) | 46468 (10%) | 85 (15%) | 0 (0%) |
| Text-line extraction | 17519 (8%) | 29140 (7%) | 35 (7%) | 65 (8%) |
| Character recognition | 32815 (15%) | 14532 (3%) | 83 (15%) | 33 (4%) |
| Total | 109701 (51%) | 101179 (24%) | 248 (46%) | 99 (11%) |

reducing the runtime by 40× compared to the previous implementation. Hence, this work results iDocChip meeting all design constraints.

## 8 Conclusion

In this paper, an optimized heterogeneous hardware-software architecture for multiresolution morphology-based *Text and Image Segmentation* is presented together with various highly optimized software implementations. Based on the new architecture, we implemented a hardware accelerator on Zynq 7045, which outperforms the original software implementation running on i7-4702MQ by a factor of 6.2 in terms of runtime and by a factor of 207 with respect to energy efficiency with negligible accuracy degradation of less than 1%. For further analysis, our hybrid hardware-software implementation for the *Text and Image*

*Segmentation* is compared with other platforms running the optimized software implementation with respect to runtime power, energy efficiency, and others. Our optimized hybrid accelerator achieves high energy efficiency and meets the objectives of iDocChip: an energy-efficient portable OCR device.

**Funding** Open Access funding enabled and organized by Projekt DEAL.

## References

1. ABBYY. https://www.abbyy.com/en-eu/. Accessed 24 Apr 2020
2. Omnipage. https://www.kofax.com/Products/omnipage?source=nuance. Accessed 24 Apr 2020
3. OCRopus. https://github.com/tmbarchive/ocropy. Accessed: 2020-04-24
4. Tesseract. https://github.com/tesseract-ocr. Accessed 24 Apr 2020
5. Bukhari, S. S., Kadi, A, Jouneh, M. A., Mir, F. M., Dengel, A: anyocr: An open-source ocr system for historical archives. In: 2017 14th IAPR international conference on document analysis and recognition (ICDAR), vol. 1 , pp. 305–310. IEEE, (2017)
6. Narrenschif. http://kallimachos.de/kallimachos/index.php/Narragonien. Accessed 24 Apr 2020
7. Tekleyohannes, M. K., Rybalkin, V., Bukhari, S. S., Ghaffar, M. M., Varela, J. A., Wehn, N. D., Andreas: idocchip - a configurable hardware architecture for historical document image processing: multiresolution morphology-based text and image segmentation. In: The 6th international embedded systems symposium (IESS), (2019)
8. Brugger, C., Dal'Aqua, L., Varela, J. A., De Schryver, C., Sadri, M., Wehn, N., Klein, M., Siegrist, Michael: a quantitative cross-architecture study of morphological image processing on cpus, gpus, and fpgas. In: 2015 IEEE symposium on computer applications and industrial electronics (ISCAIE), pp. 201–206. IEEE, (2015)
9. Qasaimeh, M., Denolf, K., Lo, J., Vissers, K., Zambreno, J., Jones, P. H.: Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels. In: 2019 IEEE international conference on embedded software and systems (ICESS), pp. 1–8. IEEE, (2019)
10. Page, A., Mohsenin, T.: An efficient and reconfigurable fpga and asic implementation of a spectral doppler ultrasound imaging system. In: 2013 IEEE 24th international conference on application-specific systems, architectures and processors, pp. 198–202. IEEE, (2013)
11. Jiang, S., He, D., Yang, C., Xu, C., Luo, G., Chen, Y., Liu, Y., Jiang, J.: Accelerating mobile applications at the network edge with software-programmable fpgas. In: IEEE INFOCOM 2018-IEEE conference on computer communications, pp. 55–62. IEEE, (2018)
12. Bonamy, R., Bilavarn, S., Muller, F., Duhem, F., Heywood, S., Millet, P., Lemonnier, F.: Energy efficient mapping on manycore with dynamic and partial reconfiguration: Application to a smart camera. Int. J. Circuit Theory Appl **46**(9), 1648–1662 (2018)
13. Xilinx, inc. zynq®-7000 All Programmable SoC. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. Accessed 24 Apr 2020
14. Baidu's apollo driverless platform. https://www.electronicdesign.com/markets/automotive/article/21119589/xilinx-soc-fpga-powers-baidus-apollo-driverless-platform. Accessed 24 Apr 2020
15. Topic embedded systems. https://topic.nl/en/products. Accessed 24 Apr 2020
16. AXIOM beta: a professional digital cinema camera. https://apertus.org/axiom. Accessed 24 Apr 2020
17. Ishikawa, S., Takahashi, T., Watanabe, S., Narukage, N., Miyazaki, S., Orita, T., Takeda, S., Nomachi, M., Fujishiro, I., Hodoshima, F.: High-speed x-ray imaging spectroscopy system with zynq

soc for solar observations. Nucl. Instrum. Methods in Phys. Res. Sect. A: Accel, Spectrometers, Detectors and Associated Equipment **912**, 191–194 (2018)
18. Mata-Carballeira, Ó., Gutiérrez-Zaballa, J., del Campo, I., Martínez, V.: An fpga-based neuro-fuzzy sensor for personalized driving assistance. Sensors **19**(18), 4011 (2019)
19. Guo, K., Sui, L., Qiu, J., Jincheng, Yu., Wang, J., Yao, S., Song Han, Yu., Wang, and Huazhong Yang., : Angel-eye: A complete design flow for mapping cnn onto embedded fpga. IEEE Trans. Computer-Aided Des. Integr. Circuits Syst. **37**(1), 35–47 (2017)
20. Afroge, S., Ahmed, B., Mahmud, F.: Optical character recognition using back propagation neural network. In: 2016 2nd international conference on electrical, computer and telecommunication engineering (ICECTE), pp. 1–4. IEEE, (2016)
21. Wei, T. C., Sheikh UU, Ab Rahman, A.A.H.: Improved optical character recognition with deep neural network. In: 2018 IEEE 14th international colloquium on signal processing and its applications (CSPA), pp. 245–249. IEEE, (2018)
22. Clausner, C., Hayes, J., Antonacopoulos, A., Pletschacher, S.: Creating a complete workflow for digitising historical census documents: considerations and evaluation. In: Proceedings of the 4th international workshop on historical document imaging and processing, pp. 83–88, (2017)
23. Das T. K., Tripathy, A. K., Mishra, A. K.: Optical character recognition using artificial neural network. In: 2017 international conference on computer communication and informatics (ICCCI), pp. 1–4. IEEE, (2017)
24. Awel, M. A., Abidi, A. I.: Review on optical character recognition. no. June, pp. 3666–3669, (2019)
25. Moysset, B., Kermorvant, C., Wolf, C., Louradour, Jérôme: Paragraph text segmentation into lines with recurrent neural networks. In: 2015 13th international conference on document analysis and recognition (ICDAR), pp. 456–460. IEEE, (2015)
26. Murdock, M., Reid, S., Hamilton, B., Reese, J.: Icdar 2015 competition on text line detection in historical documents. In: 2015 13th international conference on document analysis and recognition (ICDAR), pp. 1171–1175. IEEE, (2015)
27. Kundu, S., Paul, S., Bera, S. K., Abraham, A., Sarkar, R.: Text-line extraction from handwritten document images using gan. Expert Syst Appl **140**, 112916 (2020)
28. Bhowmik, S., Sarkar, R., Nasipuri, M., Doermann, D.: Text and non-text separation in offline document images: a survey. Int. J. Doc. Anal. Recognit(IJDAR) **21**(1–2), 1–20 (2018)
29. Eskenazi, S., Gomez-Krämer, P., Ogier, J.-M.: A comprehensive survey of mostly textual document segmentation algorithms since 2008. Pattern Recognit. **64**, 1–14 (2017)
30. Mukarambi, G., Gaikwadl, H., Dhandra, B. V.: Segmentation and text extraction from document images: Survey. In: 2019 innovations in power and advanced computing technologies (i-PACT), vol. 1, pp. 1–5. IEEE, (2019)
31. Oyedotun, O.K., Khashman, A.: Document segmentation using textural features summarization and feedforward neural network. Appl. Intel. **45**(1), 198–212 (2016)
32. Moll, M. A., Baird, H. S.: Segmentation-based retrieval of document images from diverse collections. In: Document Recognition and Retrieval XV, volume 6815, p 68150L. International Society for Optics and Photonics, (2008)
33. Bukhari, S. S.,Al Azawi, M. I. A., Shafait, F., Breuel, T.M.: Document image segmentation using discriminative learning over connected components. In: Proceedings of the 9th IAPR International Workshop on Document Analysis Systems, pages 183–190. ACM, (2010)
34. Chowdhury, S., Mandal, S., Das, A., Chanda, B.: Segmentation of text and graphics from document images. In: Ninth international conference on document analysis and recognition (ICDAR 2007), vol. 2, pp. 619–623. IEEE, (2007)
35. Bloomberg, D. S.: Multiresolution morphological approach to document image analysis. In: Proc. of the international conference on document analysis and recognition, Saint-Malo, France, (1991)
36. Bukhari, S. S., Shafait, F., Breuel, T. M.: Improved document image segmentation algorithm using multiresolution morphology. In: Document recognition and retrieval XVIII, vol. 7874, p 78740D. International society for optics and photonics, (2011)
37. Kumar, A., Rastogi, P., Srivastava, P.: Design and fpga implementation of dwt, image text extraction technique. Procedia Comput. Sci. **57**, 1015–1025 (2015)
38. Bai, X., Shi, B., Zhang, C., Cai, X., Qi, L.: Text/non-text image classification in the wild with convolutional neural networks. Pattern Recognit. **66**, 437–446 (2017)
39. Rybalkin, V., Bukhari, S. S., Ghaffar, M. M., Ghafoor, A., Wehn, N., Dengel, A.: idocchip: A configurable hardware architecture for historical document image processing: Percentile based

binarization. In: Proceedings of the ACM symposium on document engineering 2018, p. 24. ACM, (2018)
40. Tekleyohannes, M. K., Rybalkin, V., Ghaffar, M.M., Wehn, N., Dengel, A.: idocchip-a configurable hardware architecture for historical document image processing: Text line extraction. In: 2019 International conference on reconfigurable computing and FPGAs (ReConFig), pp. 1–8. IEEE, (2019)
41. Rybalkin, V., Wehn, N., Yousefi, M. R., Stricker, D.: Hardware architecture of bidirectional long short-term memory neural network for optical character recognition. In: Proceedings of the conference on design, automation and test in Europe, pp. 1394–1399. European design and automation association, (2017)
42. Chang, A. X. M., Martini, B., Culurciello, E.: Recurrent neural networks hardware implementation on fpga. nov (2015)
43. Tekleyohannes, M. K., Weis, C., Wehn, N., Klein, M., Siegrist, M.: A reconfigurable accelerator for morphological operations. In: 2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW), pp. 186–193. IEEE, (2018)
44. Tekleyohannes, M., Sadri, M., Weis, C., Wehn, N., Klein, M., Siegrist, M.: An advanced embedded architecture for connected component analysis in industrial applications. In: Design, automation and test in Europe conference and exhibition (DATE), 2017, pp. 734–735. IEEE, (2017)
45. Multi-dimensional image processing (scipy.ndimage). https://docs.scipy.org/doc/scipy-0.14.0/reference/ndimage.html. Accessed 24 Apr 2020

## Authors and Affiliations

**Menbere Kina Tekleyohannes[1] · Vladimir Rybalkin[1] · Muhammad Mohsin Ghaffar[1] · Javier Alejandro Varela[1] · Norbert Wehn[1] · Andreas Dengel[2]**

✉ Menbere Kina Tekleyohannes
tekley@eit.uni-kl.de

Vladimir Rybalkin
rybalkin@eit.uni-kl.de

1 Technische Universitat Kaiserslautern, Kaiserslautern, Germany

2 German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany