


Article

Memory-Tree Based Design of Optical Character Recognition in FPGA

Ke Yu ¹ , Minguk Kim ² and Jun Rim Choi ^{1,2,*}¹ School of Electronic and Electrical Engineering, Kyungpook National University, Daegu 41566, Republic of Korea² School of Electronics Engineering, College of IT Engineering, Kyungpook National University, Daegu 41566, Republic of Korea

* Correspondence: jrchoi@ee.knu.ac.kr; Tel.: +82-053-950-6567

Abstract: As one of the fields of Artificial Intelligence (AI), Optical Character Recognition (OCR) systems have wide application in both industrial production and daily life. Conventional OCR systems are commonly designed and implement data computation on the basis of microprocessors; the performance of the processor relates to the effect of the computation. However, due to the “Memory-wall” problem and Von Neumann bottlenecks, the drawbacks of traditional processor-based computing for OCR systems are gradually becoming apparent. In this paper, an approach based on the Memory-Centric Computing and “Memory-Tree” algorithm has been proposed to perform hardware optimization of traditional OCR systems. The proposed algorithm was first designed in software implementation using C/C++ and OpenCV to verify the feasibility of the idea and then the RTL conversion of the algorithm was done using the Xilinx Vitis High Level Synthesis (HLS) tool to implement the hardware. This work chose Xilinx Alveo U50 FPGA Accelerator to complete the hardware design, which can be connected to the x86 CPU in the PC by PCIe to form heterogeneous computing. The results of the hardware implementation show that the system this work designed can recognize characters of English capital letters and numbers within 34.24 us. The power of FPGA is 18.59 W, which saves 77.87% of energy consumption compared to the 84 W of the processor in PC.

Keywords: Optical Character Recognition; Memory-Tree; Von Neumann; Memory-Centric Computing; computer vision; high level synthesis; Xilinx Vitis; low power; Alveo; FPGA; accelerator



Citation: Yu, K.; Kim, M.; Choi, J.R. Memory-Tree Based Design of Optical Character Recognition in FPGA. *Electronics* **2023**, *12*, 754. <https://doi.org/10.3390/electronics12030754>

Academic Editor: Alexander Barkalov

Received: 28 December 2022

Revised: 22 January 2023

Accepted: 23 January 2023

Published: 2 February 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In the 21st century, Artificial Intelligence (AI) technology has taken on a variety of forms and has developed at a very rapid rate [1]. At the same time, the amount of data and information that needs to be processed has increased greatly. In particular, the increase in image data is more significant because of the popularity of portable devices represented by smartphones. It is very important to be able to process this information efficiently. On the other hand, as a traditional method of information storing and processing, text is also a very important tool for communication and information transmission in human society. In our daily lives, text can be found everywhere, such as in documents, books, signs, etc. Optical Character Recognition (OCR) [2] systems can recognize the text of images, which is one of the most important methods of image and text processing. The OCR system utilizes optical and computer technology to read text printed on paper and automatically convert it into a format that is readable by computers and understandable by humans. As one of the important parts in the AI field, OCR systems are increasingly used in both our daily lives and in industrial production, including in license plate recognition [3–6], parcel logistics identification [7], electronic medical records [8], blind navigation [9], automatic driving [10], human–computer interaction [11] and so on.

Typically, as shown in Figure 1, an OCR system consists of four main stages, which are image pre-processing (IP), character localization (CL), character segmentation (CS) and

character recognition (CR). The IP stage is a pixel pre-processing of the input image, such as image graying, image filtering and image binarization, in order to reduce the image noise and enhance the effective image information. The CL stage is to find the position of the character in the image after pre-processing in the IP stage and record its coordinate information. The CS stage is to separate the characters to be recognized from the original image to form a series of single images based on the character positions detected in the CL stage. The CR stage compares the segmented image from the CS stage with the standard template image to obtain recognition results. Among the four stages of the whole OCR system, the CR stage is the most important core and is essential to the OCR system. The computation of the CR stage will most directly affect the processing results of the entire OCR system. Thus, it can be said that the previous three stages are used to provide services for the CR stage.

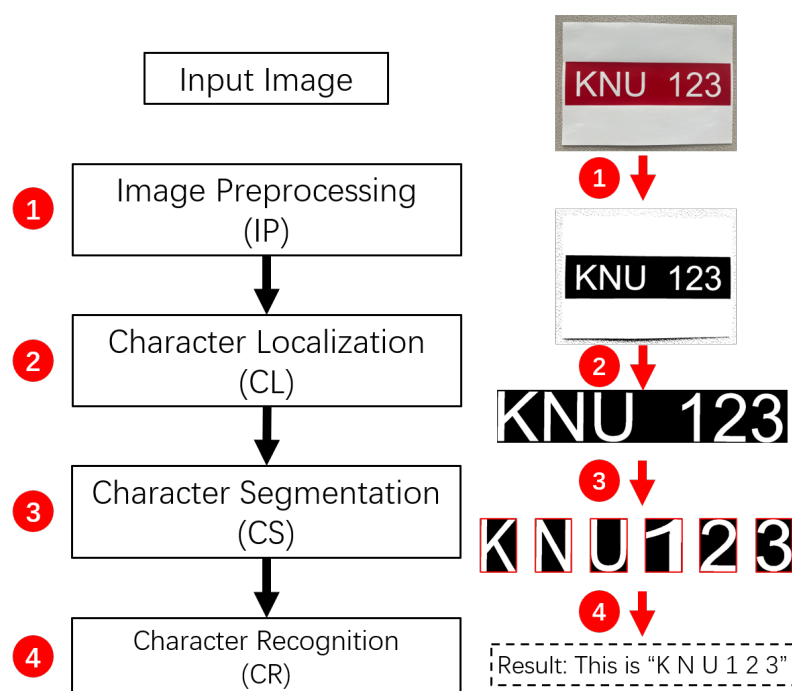


Figure 1. Four stages of the Optical Character Recognition (OCR) system for the image "KNU 123".

Traditional OCR systems are usually designed based on processors and to obtain better computational effects, it is required to use more advanced processors in terms of hardware implementation. Because the processor is the core of the computing system in the current Von Neumann Architecture [12], both the computation of data and the control of logic signals need to be finished by the processor. However, the processor has the disadvantage of high power consumption and high cost. On the other hand, advances in AI and high-definition (HD) camera technology have led to a greatly increasing amount of data in the images that computing systems need to process. So in the traditional computing architecture, computational problems such as Von Neumann bottleneck and memory wall are becoming more and more obvious. Therefore, in recent years, there has been a growing voice for developing AI application products based on Memory-Centric Computing, which uses non-Von Neumann architecture optimization [13–15].

In this paper, based on the Memory-Centric Computing principle, a new OCR system for the character recognition of numbers and English capital letters using the "Memory-Tree" algorithm optimization method was developed. The optimization design is mainly aimed at the CR stage of an OCR system. In the design of the Memory-Tree algorithm, characters with similar shapes were put into a group, such as "B" and "D", "F" and "P". When the character that needs to be recognized enters the CR stage of the operation, the system first distinguishes which group the input character belongs to and then distinguishes which

character belongs within that group. For the proposed optimization method of Memory-Tree, this work verified it using software implementation and hardware implementation, respectively. In software implementation, C/C++ language and OpenCV library were used to design a specialized software applications to verify the feasibility of the new algorithm on the software side. In hardware implementation, this work used the Xilinx Alveo U50 data center accelerator, which is a High-performance FPGA consisting of two important modules: Programmable Logic (PL) and High Bandwidth Memory (HBM) [16]. It can be connected to the x86 CPU in the host of PC over PCIe to form heterogeneous computing [17]. The proposed OCR system can recognize characters in less time and with lower power consumption through the hardware optimization design.

This paper contains the following contributions:

- This paper analyzes the problems faced by current OCR systems and traditional computing architectures.
- This paper proposes a new OCR optimization algorithm called Memory-Tree based on the principle of Memory-Centric Computing.
- Through a series of experiments in software implementation and hardware implementation, the feasibility and optimization effects of the proposed Memory-Tree algorithm are verified.

The remaining sections of this paper are organized as follows: Section 2 presents the background and related work. The Memory-Tree algorithm is introduced in Section 3. Sections 4 and 5 describe the software implementation and the hardware implementation, respectively. Section 6 presents the results and discussions. Section 7 presents the conclusions of this paper.

2. Background and Related Work

2.1. OCR

The development of Optical Character Recognition (OCR) technology began in the late 1920s, and was first proposed and patented in 1929 by the German scientist Tausheck [18]. After decades of development, OCR has become one of the most important research directions in the field of pattern recognition today.

In Section 1, the OCR system is divided into four stages; the first three stages can also be collectively referred to as the recognition preparation (RP) stage. The processing workload of the RP stage is usually unfixed and its data size and computational volume need to be determined by the actual recognition environment and requirements.

The target of the RP stage is to provide aspects to the character recognition (CR) stage processing and the CR stage is the core of the whole OCR system. Usually, there is the Template Matching Method and the Feature Extraction Method, two main approaches to implementation of the CR.

2.1.1. Template Matching Method

The Template Matching Method is a simple and straightforward traditional character recognition method, and is a good choice for when image noise is low. In [19], a Template Matching Method for recognition of Arabic numerals 0 to 9 is proposed. It performs a direct pixel similarity comparison between the binarized character input image and the standard template image, and uses a specific similarity algorithm to calculate the degree of similarity between them. The template with the highest similarity will be recognized as the target. However, The disadvantages of the Template Matching Method are also obvious in practice. When the image noise is high and the recognition is complex, the recognition success rate of the Template Matching Method decreases and is very computationally intensive [3].

2.1.2. Feature Extraction Method

The Feature Extraction Method is a recognition method based on the inherent characteristics of the character itself, which can avoid processing all pixels of the image and thus can improve the computational efficiency. In [20], a recognition method featuring

the number of intersections of characters with horizontal and vertical coordinate axes is proposed. The work in [21] proposed a method to recognize characters by building features through vertical and horizontal projections. The Feature Extraction Method is more efficient in some cases and it can even recognize character images in the presence of distortion or noise.

2.2. Von Neumann Bottleneck and Memory Wall

OCR systems typically use a combination of stand-alone scanning devices and computing devices. A traditional computing system mainly relies on the Von Neumann architecture shown in Figure 2 for design, which is mainly composed of a processor and memory, and these two modules exist separately. As computation is being performed, the processor needs to read data from memory using the system bus and write the data back to memory after the computation is completed.

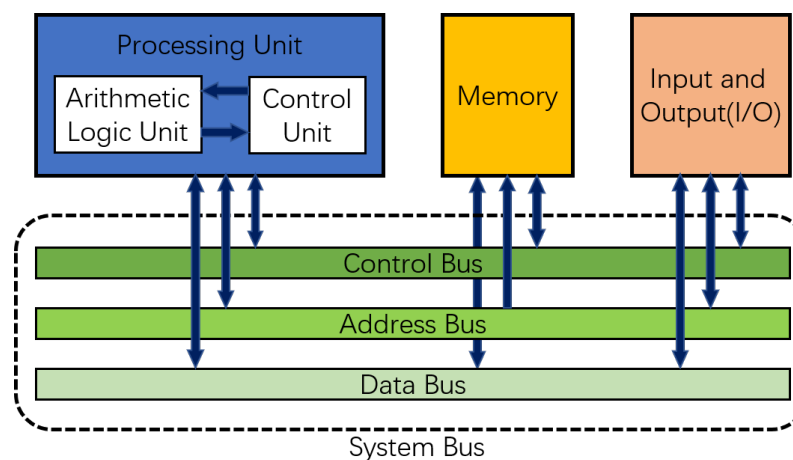


Figure 2. Architecture of Von Neumann computing system.

The emergence of Von Neumann has greatly contributed to the development of computer technology in human society. However, in the last few decades, processor performance has grown at a very high rate compared to memory due to the invention of the transistor and Moore's Law. When the memory transfer speed does not keep up with the processor's performance, it will lead to the computing power being limited as shown in Figure 3, which is known as the "Memory-Wall" problem. At the same time, as the development of the processor that relies on Moore's Law has also gradually encountered difficulties, the development of the processor itself is also slowing down.

Due to the separation of the storage and processing modules, data needs to be moved frequently between them using the system bus, resulting in a lot of wasted energy. In a study of Google's own products' energy consumption in 2018, it was found that 62.7% of the entire system's energy consumption was wasted on CPU and memory read and write transfers [22]. With the development of AI and the increase in data, this problem will be more serious. These issues have negatively impacted the development of computing systems.

2.3. Memory-Centric Computing

Memory-Centric Computing is considered a direction to solve the power and bottleneck problems of the Von Neumann architecture [13,14], which has three main technology roadmaps as shown in Figure 4b–d [15].

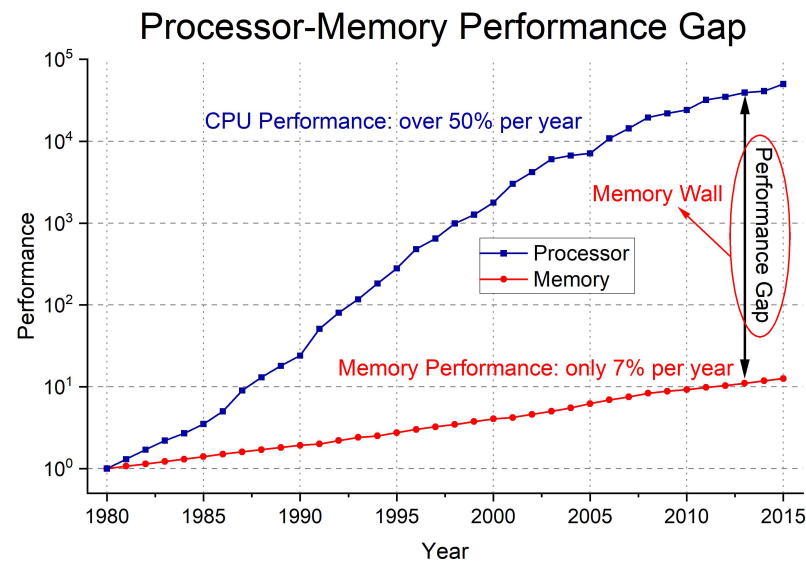


Figure 3. Performance gap of processor and memory from 1980s.

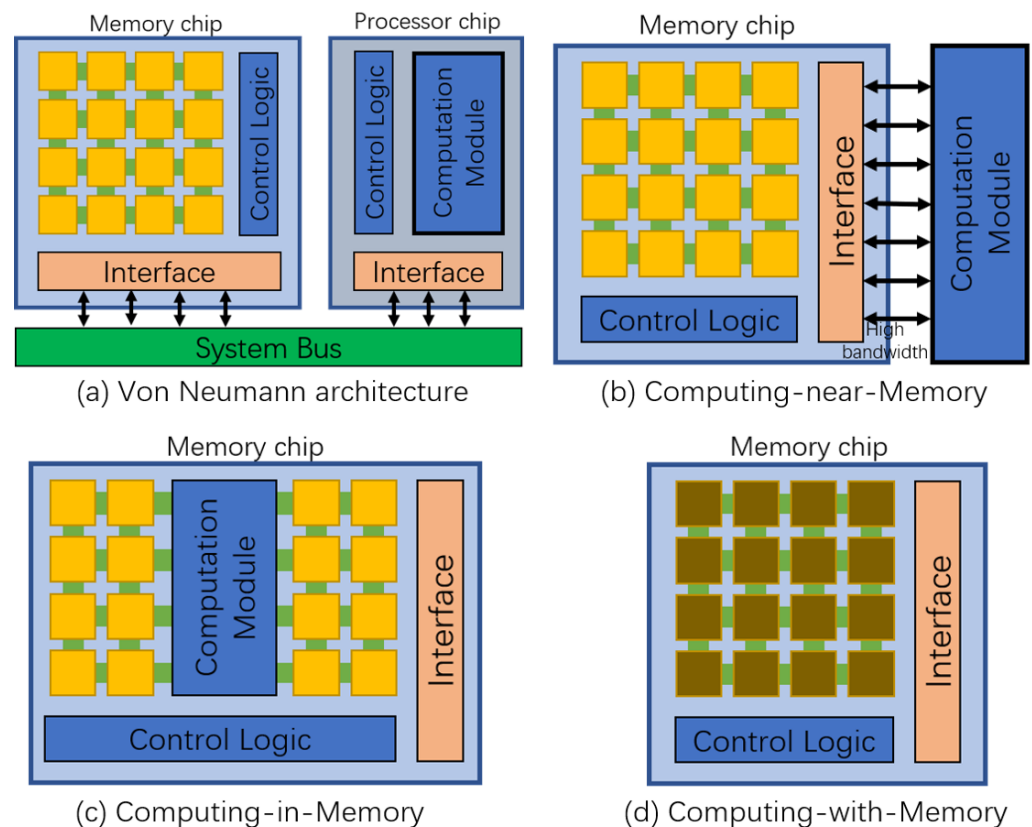


Figure 4. Comparison of Von Neumann architecture and three technology roadmaps for Memory-Centric Computing design.

2.3.1. Computing-Near-Memory (CNM) & Computing-in-Memory (CIM)

CNM [23] and CIM [24], shown in Figure 4b,c, are the continuation of the von Neumann architecture. Their principle is to place the computational unit and memory as close to one another as possible or to design the computational unit directly in the memory chip. This reduces the distance of data moving between the computation unit and storage unit and achieves the purpose of reducing power consumption and increasing computation speed.

2.3.2. Computing-with-Memory (CWM)

Different from the CNM and CIM principles, CWM shown in Figure 4d, has no separate computation unit and the computation operation is performed by the storage unit inside the memory chip. CWM requires the developer to pre-calculate the results of data processing during hardware design, and function response is stored in memory arrays in the form of Look-Up Tables (LUTs) [25]. During the computation, the function is evaluated by retrieving the values from the LUTs to complete the computation and confirm the result. CWM computations can follow either a spatial or a temporal computation model and are typically implemented in hardware using Content Addressable Memory (CAM) [26] and Field Programmable Gate Arrays (FPGAs) [27].

3. Memory-Tree Algorithm

3.1. Application of Memory-Centric Computing

Designing the special architecture of the storage module through the result of performing pre-calculation and using it to complete the calculation gives the developer an optimization idea based on Memory-Centric Computing. As an example, for calculating the diameter of a ball, people can use a computing tool such as a high-precision ruler with their brains to do the calculation and they can soon get a more accurate result. However, to put it another way, people can also perform a storage tool-based computing method using a specific transparent container designed as shown in Figure 5 by pre-calculation. People first put the ball into the designed container and due to gravity, the ball will keep falling in the container. When the ball is stuck in a certain position, people can determine the diameter of the ball according to the position it is in.

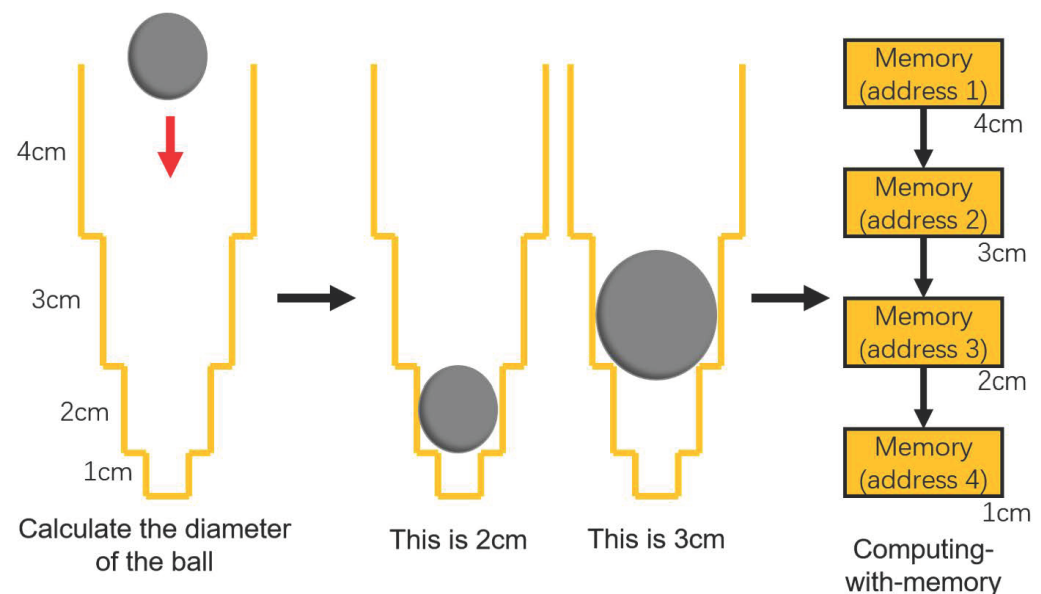


Figure 5. Example of using a container to calculate the diameter of a ball based on the Memory-Centric Computing principle.

The first method is obviously more suitable for calculations when the number of balls is small, but when the number is large and the accuracy requirement is low, it is clearly more appropriate to use the second method. The role of the ball in the example is similar to the data in a computer system. By changing the architecture of multiple storage units and controlling them appropriately, it is possible to make the storage tool perform certain computational functions, which can effectively reduce the workload of the computing tool.

3.2. Memory-Tree Algorithm in OCR System

In the introduction section, this paper introduces the importance of the CR stage, while Section 2 also introduces some algorithms commonly used in the CR stage. Moreover, the input image with characters needs to be compared with the standard character template several times at this stage, so it is speculated that the data will move more between the processor and memory at this stage. At the same time, from a hardware design perspective, the size of the data in this stage is more stable and the data movement is more predictable and controllable compared to the other stages of the OCR system. These conditions are favorable for optimizing the architecture design of the hardware. Therefore, this work combines the ideas of Sections 2.3 and 3.1 and designs a Memory-Tree optimization algorithm based on Memory-Centric Computing principles using pre-calculation.

In the “Memory-Tree” optimization algorithm, characters with similar shapes are grouped into one group. When the input image containing characters enters the CR stage, it will first distinguish which group the characters in the input image belong to. Then, the input image will be placed into the group to which it belongs, distinguishing which character it belongs to within the group. Two core computer vision algorithm functions need to be used in this process, which are “Crop” and “Sum”. The crop function is used to extract the region of interest (ROI) [28] from the input image and the sum function is used to calculate the sum of all pixels in the input image.

When recognizing characters in the CR stage, the input image and the standard template image first need to be resized to the same fixed size. As shown in Figure 6, when distinguishing the characters of “B, D, F and P”, “B” and “D” can be set together as a single group, since their pixels in position ROI 1 are almost the same. Similarly, “F” and “P” will also be set as the same group. When distinguishing the character of input image, the sum of pixel values at the ROI 1 position will be calculated first by using the crop and sum algorithm function in the computer vision library. When the sum of pixel values satisfies Equation (1), it will determine that the input image belongs to the group of “B and D”. Conversely, it will determine that the input image belongs to the group of “F and P”. When the input image belongs to the group “B and D”, the sum of pixel values at ROI 2 will be calculated. If Equation (1) is met, the input image is determined to be character “B”. Otherwise, the input image is character “D”. Similarly, the pixel value sum of ROI 3 is calculated and the input image is determined as character “P” when Equation (1) is satisfied; otherwise, the input image is determined as character “F”. The above is the central operating principle of the Memory-Tree algorithm.

$$\text{sum of ROI} \geq \text{cow(ROI)} \times \text{row(ROI)} \times 255 \times 75\% \quad (1)$$

This work designed the memory and path for CR stage processing according to the algorithm for the input image data to be moved and stored transiently in it. When the input image data are moved to a certain memory, it can be known which character group or character the picture belongs to by the address of that memory. Thus, the CR stage computation can be completed by moving the data and confirming the address of the memory where the data are located. From the software point of view, the Memory-Tree algorithm avoids repeatedly calculating all the pixels of the image and only needs to calculate the pixel sum of the crop region. Moreover, through tree architecture optimization, the algorithm effectively reduces the maximum number of comparisons in the recognition process. When the application of optimization algorithm has been extended from “B, D, F and P” to the range of numbers and English capital letters according to the principle of Memory-Tree, the architecture as shown in Figure 7 can be obtained.

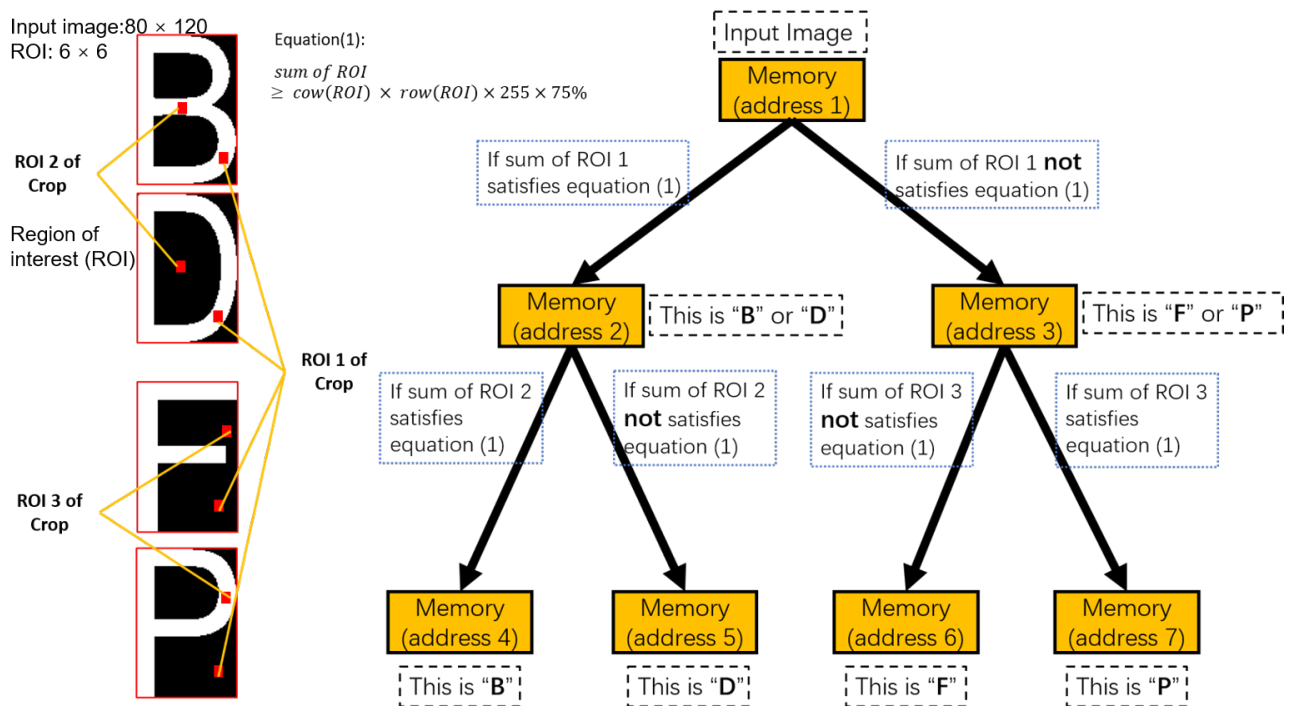


Figure 6. Description of the core principle of the Memory-Tree optimization algorithm in the character recognition (CR) stage with the example of distinguishing “B”, “D”, “F” and “P”.

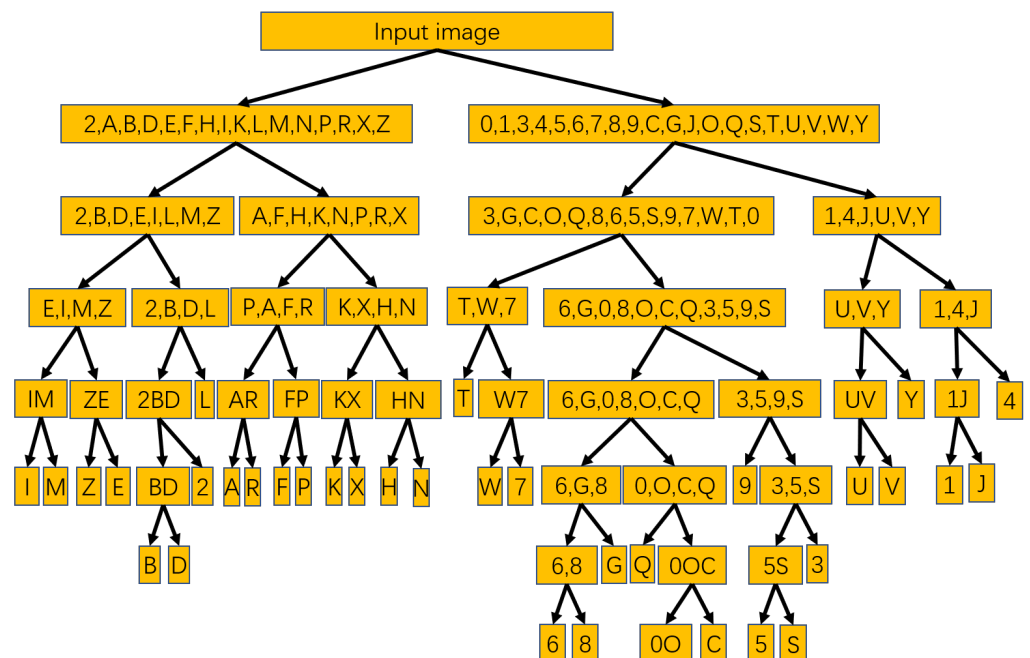


Figure 7. The architecture of the Memory-Tree algorithm applied for the recognition of 35 characters, including numbers and English capital letters. Note that “O” and “0” are considered as the same character because they are too similar.

In Sections 4 and 5, this paper verifies the feasibility and optimization of the algorithm using software implementation and hardware implementation, respectively.

4. Software Implementation with C/C++ and OpenCV

In software implementation, this work first verified the feasibility of the proposed Memory-Tree algorithm and the processing effect of the algorithm on the software side.

Usually, the software implementation of the algorithm is available in MATLAB, Python and C/C++. Considering that the C/C++ language has the best relationship with the hardware, the C/C++ approach was chosen. This work developed the proposed algorithm in C/C++ using the functions of computer vision from the OpenCV library.

The performance of the CR stage in an OCR system has been tested after development with C/C++ and OpenCV using Memory-Tree algorithm optimization. Nearly 40 characters containing a single Arabic numeral or English capital letter were randomly tested by disrupting their order. The characters were all correctly recognized and the processing times for individual characters are shown in Figure 8a. It is worth noting that the computer hardware devices used for software implementation tests are Intel i7-4790 CPU and 16 GB RAM with Ubuntu 18.04. The power consumption and frequency of CPU are 84 W and 3.60 GHz, respectively. The character image pixel size (columns \times rows) used in the development and testing of the CR stage software implementation is 80×120 and the ROI size of the crop is 6×6 . This size was chosen after several practical tests. At this size, the operation of the CR stage can balance high performance and 100% recognition accuracy at the same time.

For comparison, this work also developed an application using C/C++ and OpenCV by the traditional algorithm of the Template Matching Method, and tested it under the same conditions. The algorithm compares the binarized character input image with the standard template image one by one according to the pixel positions, and when the pixel identity exceeds 90%, the input image will be recognized as the character corresponding to that standard template image. The results of the test are shown in Figure 8b. It can be found that the test results using the traditional algorithm are significantly worse than those of the Memory-Tree algorithm.

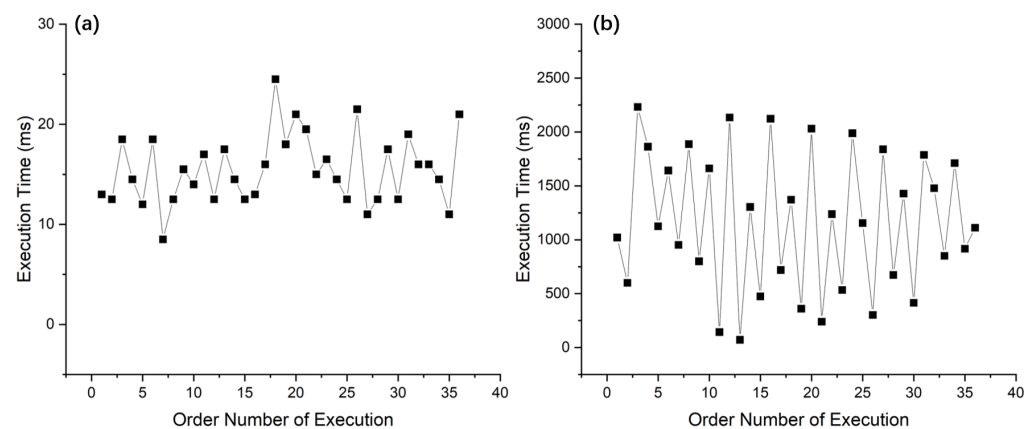


Figure 8. Execution time of the CR stage using (a) Memory-Tree algorithm optimization and (b) the Traditional Template Matching Method algorithm in software implementation.

5. Hardware Implementation with Alveo U50 and Xilinx Platform

The results of the C/C++ and OpenCV software implementations show that the Memory-Tree algorithm performs better on the software side, so a hardware implementation of it will be performed. This work used the Xilinx Alveo U50 Accelerator as the hardware platform, which is a high-performance FPGA consisting of Programmable Logic (PL) with UltraScale+ architecture and 8 GB of High Bandwidth Memory (HBM) with 32 AXI interfaces. The accelerator can be connected to the x86 CPU in the host PC through the PCIe interface, constituting a heterogeneous computing in combined Processing System (PS) and Programmable Logic (PL) mode.

Under the Xilinx development platform, this work implemented the CR stage of the OCR system in hardware based on the Memory-Tree algorithm in PL form, while the other three stages of the OCR system and signal processing parts were implemented in hardware by PS form. With the PCIe physical interface connection and communication transfer of OpenCL and AXI, it constitutes a heterogeneous computing model combining

Memory-Centric Computing and traditional computing. Figures 9 and 10 show the Xilinx Alveo U50 Accelerator installed in the host computer and the architecture of Xilinx Vitis development platform in hardware implementation.

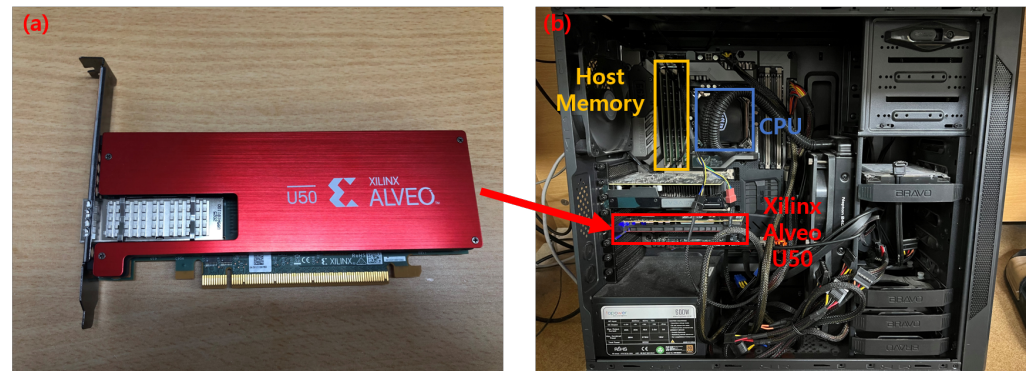


Figure 9. (a) Xilinx Alveo U50 Accelerator and (b) Accelerator installed in the host computer.

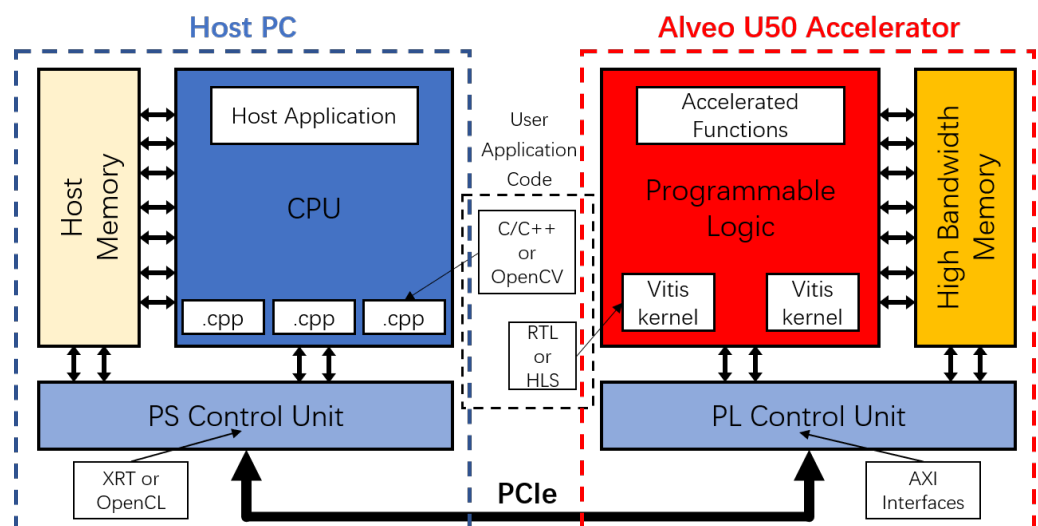


Figure 10. Architecture of Xilinx Vitis development platform and devices in hardware implementation.

5.1. PS

The implementation of the PS part needs to be done in x86 CPU based on Linux Operating System (OS). In development of this work, the PS implementation consists of two main parts, one is using the Open Computing Language (CL) [29] code for data transfer and control of each hardware on the platform and the other is using functions implemented in C/C++ and OpenCV library.

OpenCL is a commonly used programming language for heterogeneous platforms. Its main function is to use the Xilinx Runtime (XRT) driver to send the processed data from the PS to the Alveo U50 FPGA accelerator for PL processing. After the accelerator PL processing is complete, OpenCL needs to read back the result of its processing. In addition, since some of the algorithms of the OCR system are difficult to implement completely in hardware form, this work uses C/C++ functions and OpenCV libraries to complete the implementation of the IP, CL and CS stages in the same way as in the software implementation. The processed results will be sent by OpenCL to the PL for further processing in the CR stage.

5.2. PL

In PL implementation, the CR stage of the OCR system by the Memory-Tree optimization algorithm in hardware was implemented and successfully ran in the Alveo U50 Accelerator. During the development and implementation, this work used the High Level

Synthesis (HLS) [30,31] and Integrated Design Environment (IDE) [32] tools of the Xilinx Vitis unified software platform.

Xilinx offers a Vitis Accelerated Library that can be implemented in hardware and contains several computer vision functions for performing image processing [33]. The libraries use C/C++ for hardware description and can be synthesized in hardware with HLS into “Vitis Kernel” which can run in Xilinx FPGAs. This work uses the “xf::cv::crop”, “xf::cv::sum” and “xf::cv::Mat” libraries to execute the crop, sum and mat functions.

This work completed the hardware design of the Vitis Kernel using the crop, sum and mat function libraries several times repeatedly according to the architecture of the Memory-Tree algorithm from Figure 7 and defined the data form and size. The form and size of the image data are exactly the same as those used by the Memory-Tree optimization algorithm during the software implementation. After completing the hardware design, the hardware synthesis and implementation were performed, respectively, by HLS and IDE under Ubuntu 18.04 OS according to the Xilinx guidelines.

Figures 11 and 12 show the device map and power consumption information of the Alveo U50 Accelerator in hardware implementation. Table 1 shows the performance estimates for the PL hardware implementation of the Memory-Tree algorithm in CR stage and Table 2 shows its hardware utilization.

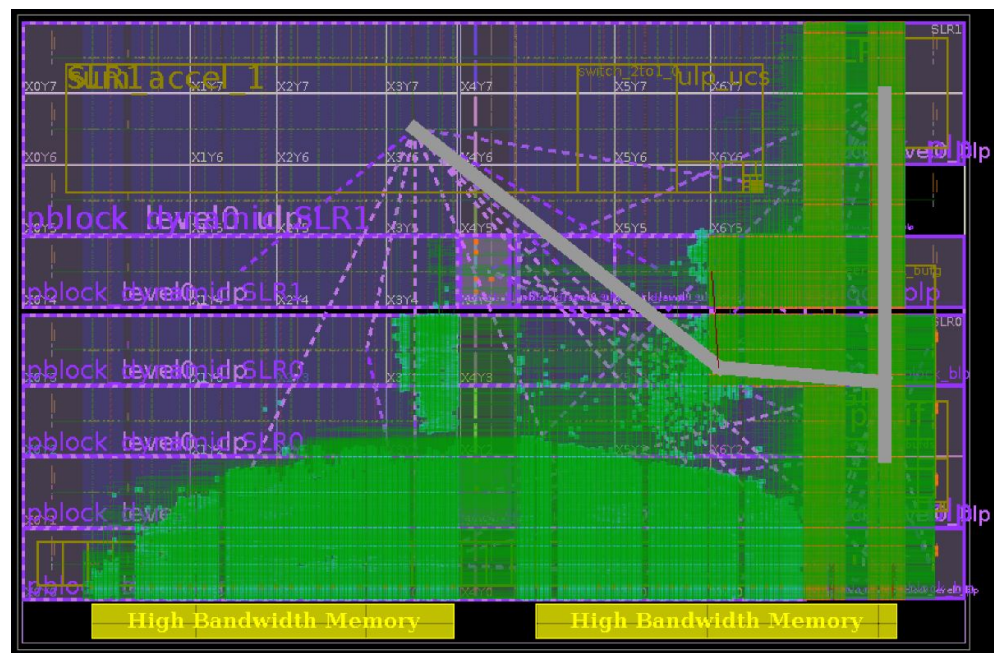


Figure 11. Device map and routing of Alveo U50 Accelerator during the hardware implementation for Memory-Tree optimization.

Table 1. Performance estimates for the PL hardware implementation.

Frequency of FPGA	Latency (Cycles)	Execution Time
300.300 MHz	10,273	34.24 us

Table 2. Utilization for the PL hardware implementation.

Name	BRAM_18K	DSP	FF	LUT	URAM
Total of use	234	0	36,663	72,813	0
Available	2688	5952	1,743,360	871,680	640
Utilization (%)	8	0	2	8	0

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	18.59 W
FPGA Power:	17.601 W
HBM Power:	0.989 W
Design Power Budget:	63 W
Power Budget Margin:	44.41 W
Junction Temperature:	68.9°C
Thermal Margin:	31.1°C (36.3 W)
Effective θ_{JA} :	0.8°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium
Launch Power Constraint Advisor to find and fix invalid switching activity	

On-Chip Power

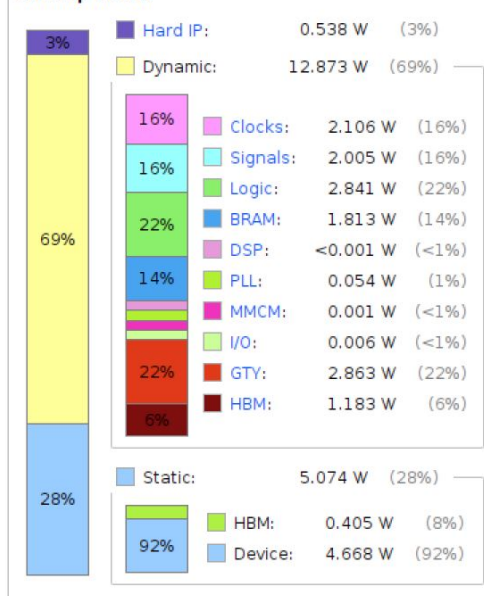


Figure 12. Power consumption results of Alveo U50 Accelerator during the hardware implementation for Memory-Tree optimization.

6. Results and Discussion

Figure 13 integrates the results of the software implementation. After calculation, it can be obtained that the traditional algorithm takes an average of 1171.97 ms to recognize a character, while the optimized algorithm takes only 15.46 ms to recognize a character. The performance improvement of the optimization algorithm in software is about 75-fold, which is a significant advantage. Since the Memory-Tree algorithm can avoid processing all pixels of the input character image and control the computation mainly on the pixel sum of the crop range, this algorithm has an advantage over the traditional algorithm purely in terms of software as well.

In terms of the results of hardware implementation, with the hardware optimization design, we can complete the recognition of a single character within 34.24 us. The hardware implementation is 451 times faster compared to the results of the software implementation. At the same time, in terms of power consumption, the CPU power used in the software implementation is 84 W, while the FPGA power used in the hardware implementation is only 18.59 W. The hardware implementation saves 77.87% of the power consumption. In other words, the Memory-Tree method achieves faster computation with lower energy consumption than the traditional CPU-centric computing method. The results of both the hardware implementation and the software implementation reflect the effectiveness of the Memory-Tree optimization.

This work compared the results of the software implementation and hardware implementation with work similar to it and the results are summarized in Tables 3 and 4. In software implementation, most researchers considered optimizing algorithms to achieve higher recognition accuracy with the smallest possible character image size. Compared to their work, the image pixel size chosen for our work is significantly larger, which led to not having a significant advantage in software execution time and also resulted in a higher FPGA resource usage. Memory-Centric Computing inherently suffers from a lack of computational accuracy, so it is reasonable to moderately increase resource usage to ensure accuracy. In fact, this work also achieves 100% accurate recognition results, which is difficult to achieve for small-sized character images. Benefiting from the advantages of FPGAs and design optimizations, the hardware implementation of this work delivers significantly better results than similar work.

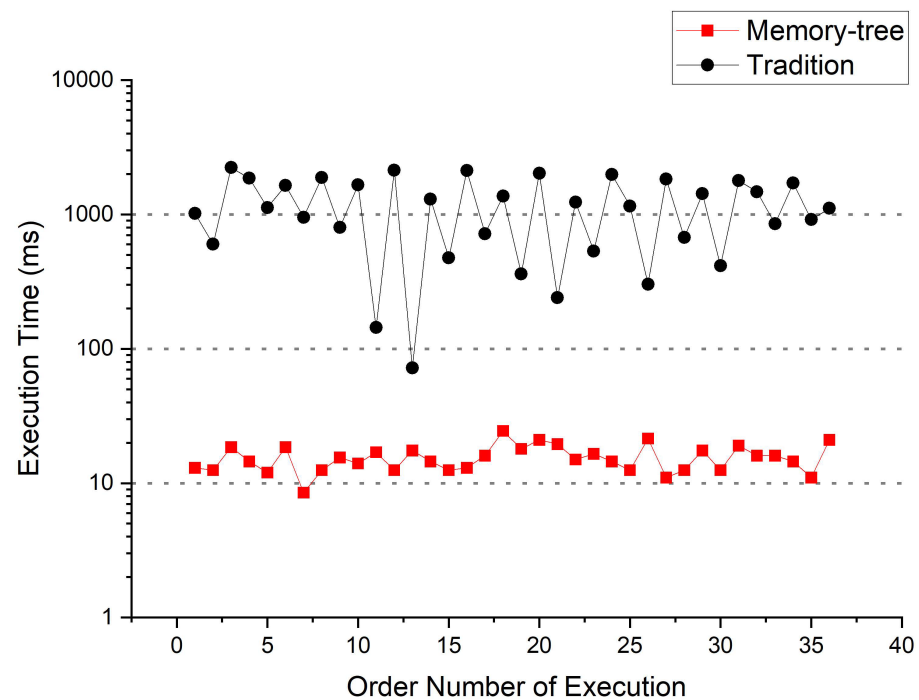


Figure 13. Comparison with Memory-Tree algorithm and traditional algorithm in software implementation.

Table 3. Comparison with similar character recognition algorithms in software implementation.

Work	Platform	Type of Characters	Pixel Size of per Character (Cols × Rows)	Processing Time of per Character
Memory-Tree	Intel i7-4790 CPU and 16 GB RAM in Ubuntu	25 English capital letters and 10 Arabic digits	80 × 120	15.46 ms
Traditional algorithm				1171.97 ms
[4]	Intel Core i7 4770s CPU and 8 GB RAM	10 Arabic digits	22 × 34	1.95 ms
[34]	i7-6700K CPU and 16 GB RAM in Ubuntu	Arabic digits	20 × 20	3.66 ms
		English capital letters		3.62 ms
[35]	ARM Cortex-A9	English capital letters and Arabic digits	9 × 19	23.40 ms
[36]	N/A	English capital letters and Arabic digits	28 × 28	N/A

However, our experimental results are more on the effectiveness of this method in terms of execution time, accuracy and power consumption. After all, data in this area are more readily available. We have insufficient data on the Memory-Tree algorithm in terms of memory and processor usage and association, which is an area we could explore more in the future. Our Memory-Tree algorithm itself also has space for further improvement. We will optimize the execution time and power consumption in our future work by trying more sizes of input images and crop ranges while the recognition accuracy can be guaranteed. We can also adjust the CPU and FPGA interface usage and function scheduling to further optimize the CPU and FPGA heterogeneous co-operation to achieve better computational results.

Table 4. Comparison with similar character recognition algorithms in hardware implementation.

Work	Memory-Tree	[4]	[37]	[35]	[38]	[39]
Platform	Xilinx Alveo U50	Zynq-7000	CME M7 FPGA	Quartus Prime 5CSEMA5F-31C6N	Xilinx Virtex IV FPGA	Altera EP1S10F484C5
Type of characters	25 English capital letters and 10 Arabic digits	10 Arabic digits	English capital letters and Arabic digits	English capital letters and Arabic digits	English capital letters and Arabic digits	Persian digits
Frequency	300.300 MHz	114.416 MHz	100 MHz	70 MHz	N/A	21 MHz
Pixel size of per character (cols × rows)	80 × 120	22 × 34	14 × 14	9 × 19	N/A	10 × 7
Processing time of per character	34.24 us	0.63 ms	6 ms	1.79 ms	N/A	47 ns
BRAM	234	10	N/A	159	N/A	N/A
DSP	0	20	N/A	18	N/A	N/A
FF	36,663	4247	N/A	1087	43,551	N/A
LUT	72,813	5616	N/A	N/A	50,310	N/A
URAM	0	N/A	N/A	N/A	N/A	N/A
Utilization	8%	6%	83.3%	N/A	N/A	8.22%

However, our experimental results are more on the effectiveness of this method in terms of execution time, accuracy and power consumption. After all, data in this area are more readily available. We have insufficient data on Memory-Tree algorithm in terms of memory and processor usage and association, which is an area we could explore more in the future. Our Memory-Tree algorithm itself also has space for further improvement. We will optimize the execution time and power consumption in our future work by trying more sizes of input images and crop ranges while the recognition accuracy can be guaranteed. We can also adjust the CPU and FPGA interface usage and function scheduling to further optimize the CPU and FPGA heterogeneous co-operation to achieve better computational results.

In addition, there is also a need to face a problem of Memory-Centric Computing, although the Memory-Centric Computing design has significant performance and power advantages. In the current conditions, with the Memory-Centric Computing method it is difficult to completely get rid of the processor alone to conduct the operation; part of the calculation and control still need to be assisted by the processor to complete the process. Therefore, it can be expected that products designed with Memory-Centric Computing will appear more and more in the coming time, but the traditional Von Neumann architecture may not be replaced anytime soon. Memory-Centric Computing will be combined with Von Neumann architecture in the form of heterogeneous computing and designed to run together for better performance through hardware and software optimization coordination.

7. Conclusions

This paper proposes a Memory-Tree hardware optimization method based on Memory-Centric Computing for running an OCR system. This method was first implemented in software using C/C++ and OpenCV and then using the Xilinx Alveo U50 Accelerator for the hardware implementation. This method enables PL+PS heterogeneous computing with x86 CPU using the Vitis unified software platform during hardware implementation. The results of the experiments show that Memory-Tree hardware optimization can achieve accurate recognition of numbers and English capital letters within 34.24 us with 18.59 W power. Compared to traditional 84 W processor computing, it saves 77.87% of power consumption and computes 451 times faster. This study verifies the feasibility and advantages of Memory-Centric Computing in terms of computational efficiency and provides a reliable

reference for research to solve the traditional Von Neumann computing bottleneck and Memory-Wall problems.

Author Contributions: Conceptualization, J.R.C.; methodology, K.Y. and M.K.; software, K.Y. and M.K.; validation, K.Y., M.K. and J.R.C.; formal analysis, K.Y. and M.K.; investigation, K.Y.; data curation, K.Y.; writing—original draft, K.Y.; writing—review & editing, K.Y. and J.R.C.; visualization, J.R.C.; supervision, J.R.C.; project administration, J.R.C.; funding acquisition, J.R.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Samsung Electronics Co., Ltd, National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R111A3065961) and BK21 Four project funded by the Ministry of Education, Korea (No. 4199990113966).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Negi, A.; Rajesh, K. A Review of AI and ML Applications for Computing Systems. In Proceedings of the 2019 9th International Conference on Emerging Trends in Engineering and Technology—Signal and Information Processing (ICETET-SIP-19), Nagpur, India, 1–2 November 2019; Volume 6, pp. 1–6.
2. Smith, R. An Overview of the Tesseract OCR Engine. In Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), Curitiba, Brazil, 23–26 September 2007; pp. 629–633.
3. Du, S.; Ibrahim, M.; Shehata, M.; Badawy, W. Automatic license plate recognition (ALPR): A state-of-the-art review. *IEEE Trans. Circuits Syst. Video Technol.* **2013**, *23*, 311–325. [\[CrossRef\]](#)
4. Farhat, A.; Hommos, O.; Al-Zawqari, A.; Al-Qahtani, A.; Bensaali, F.; Amira, A.; Zhai, X. Optical Character Recognition on heterogeneous SoC for HD automatic number plate recognition system. *EURASIP J. Image Video Process.* **2018**, *58*, 1–17. [\[CrossRef\]](#)
5. Arth, C.; Limberger, F.; Bischof, H. Real-time license plate recognition on an embedded DSP-platform. In Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition, Minneapolis, MN, USA, 17–22 June 2007; pp. 1–8.
6. Jing, Y.; Youssefi, B.; Mirhassani, M.; Muscedere, R. An efficient FPGA implementation of Optical Character Recognition for License Plate Recognition. In Proceedings of the 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE), Windsor, ON, Canada, 30 April–3 May 2017; pp. 1–4.
7. Xue, W.; Li, Q.; Xue, Q. Text Detection and Recognition for Images of Medical Laboratory Reports With a Deep Learning Approach. *IEEE Access* **2020**, *8*, 407–416. [\[CrossRef\]](#)
8. Weihao, L.; Jiamin, C.; Ning, W.; Jun, S.; Weijiao, L.; Linhua, J.; Xiaodong, C. Fast segmentation identification of express parcel barcode based on MSRCR enhanced high noise environment. In Proceedings of the 2019 2nd International Conference on Safety Produce Informatization (IICSPI), Chongqing, China, 28–30 November 2019; pp. 85–88.
9. Tashk, A.; Helfroush, M.; Karimi, V. An automatic traffic control system based on simultaneous Persian license plate recognition and driver fingerprint identification. In Proceedings of the 2012 20th Telecommunications Forum (TELFOR), Belgrade, Serbia, 20–22 November 2012; pp. 1729–1732.
10. Hairuman, I.F.B.; Foong, O.M. OCR signage recognition with skew & slant correction for visually impaired people. In Proceedings of the 2011 11th International Conference on Hybrid Intelligent Systems (HIS), Melacca, Malaysia, 5–8 December 2011; pp. 306–310.
11. Takahashi, Y. Relationship between brightness illusion and recognition performance in human–computer interaction. In Proceedings of the SICE Annual Conference 2010, Taipei, Taiwan, 18–21 August 2010; pp. 375–378.
12. Backus, J. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. *Commun. ACM* **1978**, *21*, 613–641. [\[CrossRef\]](#)
13. Yao, G.; Pellizzoni, R.; Bak, S.; Yun, H.; Caccamo, M. Global Real-Time Memory-Centric Scheduling for Multicore Systems. *IEEE Trans. Comput.* **2016**, *65*, 2739–2751. [\[CrossRef\]](#)
14. Ankit, A.; Chakraborty, I.; Agrawal, A.; Ali, M.; Roy, K. Circuits and Architectures for In-Memory Computing-Based Machine Learning Accelerators. *IEEE Micro* **2020**, *40*, 8–22. [\[CrossRef\]](#)
15. Santoro, G.; Turvani, G.; Graziano, M. New logic-in-memory paradigms: An architectural and technological perspective. *Micromachines* **2019**, *10*, 368. [\[CrossRef\]](#) [\[PubMed\]](#)
16. Xilinx. Get Moving with Alveo. ug1352, 2019, Volume 1. Available online: <https://www.xilinx.com/developer/articles/acceleration-basics.html> (accessed on 1 February 2023).
17. Li, Y.; Zhao, X.; Cheng, T. Heterogeneous Computing Platform Based on CPU+FPGA and Working Modes. In Proceedings of the 2016 12th International Conference on Computational Intelligence and Security (CIS), Wuxi, China, 16–19 December 2016; pp. 669–672.

18. Mori, S.; Suen, C.; Yamamoto, K. Historical review of OCR research and development. *Proc. IEEE* **1992**, *80*, 1029–1058. [CrossRef]
19. Farhat, A.; Al-Zawqari, A.; Al-Qahtani, A.; Hommos, O.; Bensaali, F.; Amira, A.; Zhai, X. OCR based feature extraction and template matching algorithms for Qatari number plate. In Proceedings of the 2016 International Conference on Industrial Informatics and Computer Systems (CIICS), Sharjah, United Arab Emirates, 13–15 March 2016; pp. 1–5.
20. Chen, S.; Song, Q.; Guo, H.; Li, X. Design of license plate recognition system based on FPGA. In Proceedings of the 2022 2nd Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS), Shenyang, China, 25–27 February 2022; pp. 227–231.
21. Rahman, C.A.; Badawy, W.; Radmanesh, A. A real time vehicle's license plate recognition system. In Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, Miami, FL, USA, 22–22 July 2003; pp. 163–166.
22. Boroumand, A.; Ghose, S.; Kim, Y.; Ausavarungrun, R.; Shiu, E.; Thakur, R.; Kim, D.; Kuusela, A.; Knies, A.; Ranganathan, P.; et al. Google workloads for consumer devices: Mitigating data movement bottlenecks. *Assoc. Comput. Mach.* **2018**, *53*, 316–331.
23. Singh, G.; Chelini, L.; Corda, S.; Awan, A.J.; Stuijk, S.; Jordans, R.; Corporaal, H.; Boonstra, A.J. A review of near-memory computing architectures: Opportunities and challenges. In Proceedings of the 2018 21st Euromicro Conference on Digital System Design (DSD), Prague, Czech Republic, 29–31 August 2018; pp. 608–617.
24. Chen, B.; Cai, F.; Zhou, J.; Ma, W.; Sheridan, P.; Lu, W.D. Efficient in-memory computing architecture based on crossbar arrays. In Proceedings of the 2015 IEEE International Electron Devices Meeting (IEDM), Washington, DC, USA, 7–9 December 2015; pp. 17.5.1–17.5.4.
25. Courtney, T.; Turner, R.; Woods, R. Mapping multi-mode circuits to LUT-based FPGA using embedded MUXes. *IEEE Symp. FPGAs Cust. Comput. Mach. Proc.* **2002**, *2002*, 318–320.
26. Eshraghian, K.; Cho, K.R.; Kavehei, O.; Kang, S.K.; Abbott, D.; Kang, S.M.S. Memristor MOS content addressable memory (MCAM): Hybrid architecture for future high performance search engines. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2011**, *19*, 1407–1417. [CrossRef]
27. Nomani, T.; Mohsin, M.; Pervaiz, Z.; Shafique, M. XUAVs: Towards Efficient Approximate Computing for UAVs - Low Power Approximate Adders with Single LUT Delay for FPGA-Based Aerial Imaging Optimization. *IEEE Access* **2020**, *8*, 102982–102996. [CrossRef]
28. Iqbal, O.; Muro, V.I.T.; Katoch, S.; Spanias, A.; Jayasuriya, S. Adaptive Subsampling for ROI-Based Visual Tracking: Algorithms and FPGA Implementation. *IEEE Access* **2022**, *10*, 90507–90522. [CrossRef]
29. Stone, J.E.; Gohara, D.; Shi, G. OpenCL: A parallel programming standard for heterogeneous computing systems. *Comput. Sci. Eng.* **2010**, *12*, 66–72. [CrossRef] [PubMed]
30. Amiri, P.; Perard-Gayot, A.; Membarth, R.; Slusallek, P.; Leiba, R.; Hack, S. FLOWER: A comprehensive dataflow compiler for high-level synthesis. In Proceedings of the 2021 International Conference on Field-Programmable Technology (ICFPT), Auckland, New Zealand, 6–10 December 2021.
31. Xilinx. Vitis High-Level Synthesis User Guide. ug1399, 2022, Volume 2. Available online: <https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Introduction> (accessed on 1 February 2023).
32. Xilinx. Vitis Unified Software Platform Documentation: Application Acceleration Development. ug1393, 2022, Volume 2. Available online: <https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration> (accessed on 1 February 2023).
33. Valido, M.J.R.; Castello, E.M.; Medina, P.S.; González, A.M. Accelerating applications with Vitis unified environment. Case study: Vitis Vision Library. In Proceedings of the 2022 Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (XV Technologies Applied to Electronics Teaching Conference), Teruel, Spain, 29 June–1 July 2022.
34. Amirgaliyev, B.Y.; Kuvatov, K.K.; Baibatyr, Z.Y. Application of Convolutional Neural Network for Optical Character Recognition Designed for Kazakhstan Identity Cards. In Proceedings of the 2017 IEEE 11th International Conference on Application of Information and Communication Technologies (AICT), Moscow, Russia, 20–22 September 2017; pp. 1–3.
35. Sborz, G.A.; Pohl, G.A.; Viel, F.; Zeferino, C.A. A custom processor for an FPGA-based platform for automatic license plate recognition. In Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design, São Paulo, Brazil, 26–30 August 2019.
36. Raj, S.; Gupta, Y.; Malhotra, R. License Plate Recognition System using Yolov5 and CNN. In Proceedings of the 2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 7 June 2022; pp. 372–377.
37. Zho, H.; Zhu, G.J.; Peng, Y. A RMB Optical Character Recognition system using FPGA. In Proceedings of the 2016 IEEE International Conference on Signal and Image Processing (ICSIP), Beijing, China, 13–15 August 2016; pp. 539–542.
38. Caner, H.; Gecim, H.S.; Alkar, A.Z. Efficient embedded neural-network-based license plate recognition system. *IEEE Trans. Veh. Technol.* **2008**, *57*, 2675–2683. [CrossRef]
39. Toosizadeh, N.; Eshghi, M. Design and implementation of a new Persian digits OCR algorithm on FPGA chips. In Proceedings of the 2005 13th European Signal Processing Conference, Antalya, Turkey, 4–8 September 2005; pp. 1–4.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.