UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Hardware Implementation of the Otsu's Method Applied to Real-Time Worm Segmentation

Wysterlânya Kyury Pereira Barros

Advisor: Prof. Dr. Marcelo Augusto Costa Fernandes

Master's Dissertation presented to the Graduate Program in Electrical and Computer Engineering of UFRN (Concentration area: Computer Engineering) as part of the requirements for obtaining the title of Master of Science.

PPgEEC Order Number: M640
Natal, RN, November, 2021

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Barros, Wysterlânya Kyury Pereira.
Hardware implementation of the Otsu's method applied to real-time worm segmentation / Wysterlanya Kyury Pereira Barros. - 2021.
67 f.: il.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Natal, RN, 2022.
Orientador: Prof. Dr. Marcelo Augusto Costa Fernandes.

1. FPGA - Dissertação. 2. Image segmentation - Dissertação. 3. Otsu's method - Dissertação. 4. Worm tracking - Dissertação. 5. C. elegans - Dissertação. I. Fernandes, Marcelo Augusto Costa. II. Título.

RN/UF/BCZM CDU 004.41

Elaborado por Ana Cristina Cavalcanti Tinôco - CRB-15/262

Aos meus pais, João e Zilma, por todo o apoio e carinho.

Agradecimentos

Em primeiro lugar agradeço a Deus, por me dar forças nos momentos mais difíceis e perseverança para não desanimar durante a realização deste trabalho.

Aos meus pais, João e Zilma, meu porto seguro. Agradeço por todo o apoio e amor que sempre recebi, sem eles a realização desse sonho não seria possível.

Ao meu querido irmão, Kayo, pelo apoio e por ser um exemplo de determinação.

Às minhas irmãs de coração, Dayane e Mayara, pelo incentivo e apoio durante os momentos mais difíceis.

Ao meu amado companheiro, Alcemy, por toda a paciência, carinho e apoio durante a realização deste trabalho.

Ao meu orientador, Professor Dr. Marcelo Augusto Costa Fernandes, por todos os ensinamentos, incentivo e paciência. Agradeço por sua confiança e incansável dedicação.

Aos amigos e colegas, pelo apoio constante. Em especial, Daniel Morais e Mailson Ribeiro, com quem compartilhei os últimos setes anos de jornada acadêmica, pelo companheirismo e pela troca de experiências que me permitiram crescer em diversos aspectos.

Aos colegas do LAMII, por todos os momentos de descontração e pelo auxílio durante a realização deste trabalho.

A coordenação do PPgEEC e a UFRN, pelo ambiente de estudo saudável, motivador e repleto de oportunidades.

A todos os professores, por compartilharem comigo o seus conhecimentos.

À CAPES, pelo apoio financeiro.

E a todos aqueles que não foram citados aqui e colaboraram, direta ou indiretamente, com a conclusão deste trabalho, meus sinceros agradecimentos.

Abstract

Behavioral genomic studies employing the worm *Caenorhabditis elegans* have aided the discovery of new gene-behavioral associations and the screening of new drugs. High-resolution cameras record experiments with this worm, generating videos that computational solutions will later process for automated behavioral analysis. Because of the large volume of data to be processed, these analyses usually have to be performed offline. However, it is desired to develop a high-throughput implementation capable of operating in real-time, seeking to reduce the memory occupation by storing videos and allow the realization of new kinds of experiments. One way to speed up the algorithms employed is through the use of reconfigurable computing. Therefore, this work proposes the hardware development of the Otsu method for worm segmentation in real-time. The proposed implementation was developed in Field Programmable Gate Array (FPGA) using a fully parallel strategy with fixed-point representation. Architecture details are presented, as well as synthesis results related to the hardware area occupation, throughput, and dynamic power consumption. Results about validation of the implementation using images of the worms are also provided. The data show that the proposed architecture can achieve high speedups compared to similar work presented in the literature, besides allowing the segmentation of worms in real-time.

Keywords: FPGA, Image Segmentation, Otsu's Method, Worm Tracking, *C. elegans*.

Resumo

Estudos sobre a genômica comportamental empregando o verme *Caenorhabditis elegans* têm auxiliado a descoberta de novas associações gene-comportamentais e a triagem de novas drogas. Os experimentos envolvendo esse verme são gravados por câmeras de alta resolução, gerando vídeos que serão posteriormente processados por soluções computacionais para análise comportamental automatizada. Devido o grande volume de dados para ser processado, essas análises são geralmente realizadas *offline*. Contudo, deseja-se desenvolver uma implementação de alto *throughput* capaz de operar em tempo real, buscando reduzir a ocupação de memória pelo armazenamento dos vídeos e permitir a realização de novos tipos de experimentos. Uma maneira de acelerar os algoritmos empregados é através do uso de computação reconfigurável. Diante disso, este trabalho propõe o desenvolvimento em hardware do método de Otsu para a segmentação dos vermes em tempo real. A implementação proposta foi desenvolvida em *Field Programmable Gate Array* (FPGA) utilizando uma estratégia totalmente paralela com representação em ponto fixo. Os detalhes da arquitetura são apresentados, assim como resultados de síntese relacionados a área de ocupação, tempo de processamento e consumo de potência dinâmica. Também são fornecidos resultados sobre a validação da implementação utilizando imagens dos vermes. Os dados mostram que a arquitetura proposta consegue obter elevados *speedups* em comparação com trabalhos semelhantes apresentados na literatura, além de permitir a segmentação dos vermes em tempo real.

Palavras-chave: FPGA, Segmentação de Imagem, Método de Otsu, Rastreamento de Vermes, *C. elegans*.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

R^2	R-squared
BCVM	Between-Class Variance Module
BRAM	Look-Up Table
CPU	Central Process Unit
DSP	Digital Signal Processor
FN	False Negative
FP	False Positive
FPGA	Field-Programmable Gate Arrays
FPS	Frames per second
GB	Gigabyte
GPU	Graphics Processing Units
IOB	Input/Output Block
IoU	Intersection over Union
IPS	Images Processed per Second
LC	Logic Cells
LUT	Look-Up Table
Mbps	Megabits processed per second
MIM	Mean Intensity Module
MWT	Multi-Worm Tracker
NHM	Normalized Histogram Module
OTM	Optimal Threshold Module
PCNH	Partial Component of the Normalized Histogram

PCOM	Probability of Class Occurrence Module
PHENOSPACE	Quantifying behavioural phenotype space: chemistry-to-gene screens and combination therapies
PNH	Partial Normalized Histogram
R	Register
RAM	Random Access Memory
RC	Reconfigurable Computing
RTL	Register-Transfer Level
TB/h	Terabytes per hour
TN	True Negative
TP	True Positive
VHDL	Very High-Speed Integrated Circuit Hardware Description Language

Chapter 1

Introduction

Behavioral genomics aims to understand the mapping between genome variation and behavior. Several studies in this field have used the worm *Caenorhabditis elegans* (*C. elegans*), due to its relatively simple and well-characterized nervous system (Antoshechkin & Sternberg 2007, Nigon & Félix 2017, Brown et al. 2013). These studies investigate the worm's behavioral changes after chemical and genetic perturbations, seeking to discover new gene-behavior associations (Maulik et al. 2017, Evans et al. 2021) and aid in drug screening (Giunti et al. 2021, Kim et al. 2017). These analyses can occur by observing microscopic images, but performing this manually is usually exhaustive and costly. Therefore, many works have proposed the development of solutions using computer vision methods to automate this process (McDiarmid et al. 2018, O'Reilly et al. 2014, Husson et al. 2013). Thus, the analysis time of the experiments can be drastically reduced, besides providing more accurate and unbiased data (McDiarmid et al. 2018).

The developed computational solutions perform the extraction of behavioral characteristics of the worms by automatically processing videos of the experiments (McDiarmid et al. 2018, Brown & Schafer 2015). These videos are usually captured by high-resolution cameras, to provide images of the worms with a high level of detail (Barlow et al. 2021, Swierczek et al. 2011). So the recorded videos can generate terabytes of data in just a few hours, requiring a large amount of storage space and a long processing time (Barlow et al. 2021). Therefore, aiming to reduce the use of memory and allow more complex studies on behavioral genomics, the development of real-time applications has been explored. The acceleration of the algorithms employed could occur through the use of Graphics Processing Units (GPU) and Reconfigurable Computing (RC). The real-time requirements of tracking systems preclude the use of the cloud, which can offer GPU and RC resources, with the need to implement the solutions at the data acquisition location.

Various works in the literature present comparisons between GPU and RC for the acceleration of computer vision and digital image processing algorithms. In HajiRassouliha et al. (2018), a review of several papers that provide this analysis is presented, concluding that for applications that require high processing speed, RCs are better suited. The reconfigurable computing, with Field-Programmable Gate Arrays (FPGAs), allows the exploitation of the algorithm parallelization and the development of dedicated hardware to obtain performance improvement (Dias et al. 2021, Silva et al. 2020, Torquato & Fernandes 2019, Da Costa et al. 2019, Coutinho et al. 2019). In this way, the designed hardware can take advantage of the inherent parallelism of images (Vahid 2010, Gokhale

& Graham 2005).

Thereby, the implementation of the system in hardware may allow the achievement of high throughput, enabling the analysis of worms in real-time. The development of the entire framework in hardware involves the architecture design of multiple computer vision algorithms. The image segmentation algorithm is one of the most interesting to obtain real-time performance, aiming to reduce the memory occupation problem. In this step, the image pixels are divided as belonging to the worms region or the image background. Pixels classified as the background are represented by a value equal to zero. Thus, using compression algorithms, captured images could occupy less memory space as they have most pixels represented by zero. Many works in the literature use Otsu's method for worm segmentation. Proposed in Otsu (1979), this method is a widely used global thresholding technique, which proposes the definition of an optimal threshold by maximizing the between-class variance.

Thus, the present dissertation proposes a fully parallel FPGA implementation of the Otsu algorithm applied to real-time worm segmentation. This work is inserted in a broader context, linked to the project Quantifying Behavioural Phenotype Space: Chemistry-to-Gene Screens and Combination Therapies (PHENOSPACE) from Behavioral Genomics Lab of the Imperial College London, which works with the analysis of the behavior of worms with genomic changes. One of the goals of this project is to develop a high-performance image processing platform to capture complex behavioral sequences and automatically interpret them in real-time. Thus, this work contributes to the development of a dedicated high-throughput hardware framework for the real-time tracking of worms.

1.1 Dissertation Contributions

The main contribution presented is a fully parallel implementation of Otsu's method in FPGA, capable of segmenting videos with high resolution and elevated frame rate in real-time, applied to worm segmentation.

The hardware architecture was designed with a fully parallel scheme and fixed-point resolution. Through this, it was possible to reduce the processing time of the algorithm. The synthesis results were compared with other works in the literature, verifying the achievement of higher throughputs and reduction of dynamic power consumption by the proposed implementation. The high throughput achieved allowed the application of the designed hardware in the segmentation of worms in real-time.

1.2 Objectives

This dissertation aims to collaborate with the PHENOSPACE project in the development of a hardware framework for tracking worms. The proposal is associated with the hardware implementation of Otsu's method in FPGA, seeking to obtain high throughput values that allow segmentation of the worms in real-time.

The specific objectives of this work are:

- Development of Otsu's method on FPGA with a fully parallel architecture and fixed-point representation;
- Achieving high throughputs that enable real-time application;
- Obtaining high speedups by the developed hardware when compared to other works in the literature;
- Application of architecture in real-time worm segmentation.

1.3 Published Papers

1. BARROS, W. K. P. et al. Proposal of the CAD System for Melanoma Detection Using Reconfigurable Computing. *Sensors*, v. 20, n. 11, p. 3168, 2020. DOI: 10.3390/s20113168.

2. BARROS, W. K. P.; FERNANDES, M. A. C. Proposta de Implementação em Hardware do Algoritmo de Otsu Aplicado ao Rastreamento em Tempo Real de Vermes. *Anais do Computer on the Beach*, v. 12, p. 339-346, 2021. DOI: 10.14210/cotb.v12.p339-346.

3. BARROS, W. K. P.; DIAS, L. A.; FERNANDES, M. A. C. Fully Parallel Implementation of Otsu Automatic Image Thresholding Algorithm on FPGA. *Sensors*, v. 21, n. 12, p. 4151, 2021. DOI: 10.3390/s21124151.

4. TARGINO, M. T.; BARROS, W. K. P.; FERNANDES, M. A. C. Proposta de Implementação Paralela do Método Naive Bayes em FPGA. *In*: XV Simpósio Brasileiro de Automação Inteligente, 15., 2021, Online.

1.4 Dissertation Organization

This dissertation is organized in 4 chapters, as presented in the following paragraphs.

In this first chapter, an introduction was presented, in which the motivation and theme of the work are contextualized, as well as the main objectives of the proposal and published articles during the master's period.

Chapter 2 proposes a fully parallel design of Otsu's method on FPGA. The implementation details and an analysis of the synthesis results concerning the hardware area occupation, throughput, and dynamic power consumption are presented. As well as comparisons with other works in the literature.

Chapter 3 presents a study about the application of the developed hardware in real-time worm segmentation. Analyses are performed using different numbers of bits in the decimal part of the architecture and in the representation of the pixels of the image. Therefore, besides the segmentation results, synthesis results regarding the hardware area occupation and processing time are presented.

Finally, Chapter 4 presents the conclusions of this dissertation and perspectives for future works.

Chapter 2

Otsu Algorithm on FPGA

This chapter describes a high-throughput implementation of the Otsu algorithm on FPGA, aiming to segment high-resolution videos in real-time. The architecture designed uses a fully parallel strategy associated with fixed-point resolution. This chapter presents the implementation details, as well as synthesis results related to the hardware area occupation, throughput, and dynamic power consumption. These results are used to evaluate and compare the performance of this proposal with other works in the literature.

2.1 Introduction

The Otsu algorithm, used in worm tracking systems, has a high computational cost due to the complex arithmetic operations performed interactively, hindering its use in real-time applications. Therefore, many studies proposed the Otsu algorithm developed in hardware, such as FPGA, to overcome the processing time constraints. This allows the achievement of real-time or near real-time processing. However, FPGA implementations found in the literature are often developed with sequential processing schemes in some stages of the Otsu algorithm, limiting the hardware's processing speed (Jianlai et al. 2009, Tian et al. 2003, Pandey et al. 2014, Torres-Monsalve & Velasco-Medina 2016, Pandey & Karmakar 2019, Das et al. 2014).

Many proposals can be found in the literature for real-time applications of the Otsu algorithm deployed in FPGAs. In Wang & Huang (2013), an adaptive lane departure detection and alert system is presented, while in Zhao et al. (2014), a lane departure and frontal collision warning system. Meanwhile, in Alhamwi et al. (2015), a vision system is presented to detect obstacles and locate a robot that navigates indoors; in Ren & Wang (2016), is presented a system for detecting moving objects; Tulasigeri & Irulappan (2016), presents a system to assist in the diagnosis of glaucoma; and, in Kim (2018), a system for improving thermograms. However, these articles provide few details about the hardware implementation.

Among the first FPGA implementations of the Ostu algorithm is the proposal of Jianlai et al. (2009), synthesized for an Altera Cyclone II FPGA. The design improved the algorithm's performance through a hybrid hardware architecture and Altera MegaCores, eliminating complex divisions and multiplications of the algorithm. The architecture developed was used for the segmentation of an image with a resolution of 320×240 and

pixel represented by 10 bits. Through visual segmentation results, they evaluated the implementation performance as satisfactory.

Other works have proposed a hardware implementation using logarithmic functions to eliminate the division and multiplication circuits. In Tian et al. (2003) and Torres-Monsalve & Velasco-Medina (2016), the authors implemented two versions of the Otsu algorithm, in which one version uses the logarithmic functions, to compare the results achieved between them. The architectures developed by Tian et al. (2003) were synthesized for a Xilinx Virtex XCV800 HQ240-4 FPGA. The implementation without the logarithmic functions occupied a hardware area of 622 slices and 103 Input/Output Blocks (IOBs), obtaining a clock latency of 362.4ns. Meanwhile, the logarithmic function implementation occupied 109 slices and 49 IOBs, obtaining a clock latency of 132ns.

The architectures presented by Torres-Monsalve & Velasco-Medina (2016) were developed on the Altera Cyclone IV EP4CE115F29C6N FPGA. The synthesis results obtained for the algorithm implemented without the logarithmic functions occupied $6,525$ logic elements, $4,920$ registers, $18,266$ bits of memory, and 79 multipliers of 9 bits. Regarding the processing time, considering an image with a resolution of $1,280 \times 1,024$, the system achieved a maximum frequency of 130.89 MHz and latency of 589 clock cycles. In contrast, the implementation with the logarithmic function used $2,440$ logic elements, $1,026$ registers, $10,943$ bits of memory, and 79 multipliers of 9 bits. Moreover, it achieved a maximum frequency of 114.62 MHz and a latency of 536 clock cycles. Therefore, the results presented in Tian et al. (2003), and Torres-Monsalve & Velasco-Medina (2016) indicate that the algorithm designed with logarithmic function reduced the FPGA area occupation and latency.

In Pandey et al. (2014) and Pandey & Karmakar (2019), were deployed similar architectures of the Otsu algorithm in the Virtex-5 xc5vfx70t ffg1136-1 FPGA, available on the Xilinx ML-507 development platform. Both proposals were developed in VHDL, using fixed-point representation, operating at a clock frequency of 25.175MHz. The proposal described in Pandey et al. (2014) occupied 168 slices and 33 IOBs, while in Pandey & Karmakar (2019), the implementation reached an area occupation of 161 slices, 21 IOBs, 72 Look-Up Tables (LUTs), 591 registers, 4 blocks of RAM (BRAMs) and 5 DSP48Es. In addition, Pandey & Karmakar (2019) presented results related to the processing time for a 640×480 image, with pixels represented by 8 bits. This work reached a latency of 5 clock cycles and throughput of 40.28 megabits processed per second (Mbps).

Meanwhile, it was presented in Das et al. (2014) an implementation for binarization and thinning of fingerprint images, using the Otsu method, on a Spartan 6 LX45 FPGA. Concerning the area occupation, $1,898$ registers, $1,859$ LUTs, 735 slices, 10 IOBs and 44 BRAMs were used. Regarding processing time, a maximum clock frequency of 100MHz was achieved, with the execution time of $1,489$ms for processing a 280×265 image and latency of 531 clock cycles or $5,310$ns. Besides, a comparison with the same technique implemented in Matlab was also presented, showing that the FPGA was $\approx 10\times$ faster than the Matlab version.

Thus, this chapter presents a fully parallel FPGA implementation of the Otsu algorithm. Unlike most approaches proposed in the literature, a full-parallel implementation reduces the bottleneck for processing speed compared to sequential systems or hybrid

hardware architectures, that is, architectures implemented with sequential and parallel schemes. The hardware implementation was developed in *Register-Transfer Level* (RTL), using fixed-point representation, in an Arria 10 GX 1150 FPGA. The results concerning the hardware area occupation and throughput are also presented.

The remainder of this chapter is organized as follows: Section 2.2 addresses the theoretical foundation of the Otsu method; Section 2.3 shows a detailed description of the architecture proposed; while Section 2.4 present and analyze the synthesis results obtained from the described implementation, including a comparison to other works. Finally, Section 2.6 presents the final considerations about this chapter.

2.2 Otsu's Algorithm

The Otsu is one of the most popular thresholding algorithms, used to find an optimal threshold that separates an image into two classes: the background and object. These classes are represented by C_0 and C_1, respectively. This method has the advantage of performing all its calculations based only on the histogram of the image (Gonzalez & Woods 2018, Otsu 1979).

Initially, the algorithm starts by calculating the normalized histogram of an image, $\mathbf{A}$, in grayscale, which is described as

$$\mathbf{A}(m) = \begin{bmatrix} a_{0,0}(n) & \cdots & a_{0,j}(n) & \cdots & a_{0,M-1}(n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i,0}(n) & \cdots & a_{i,j}(n) & \cdots & a_{i,M-1}(n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{N-1,0}(n) & \cdots & a_{N-1,j}(n) & \cdots & a_{N-1,M-1}(n) \end{bmatrix} \tag{2.1}$$

where $a_{i,j}$ is one pixel of b bits and $M \times N$ is the image dimension. The pixels can assume L distinct integer intensity levels, represented by k and characterized as a value in a range of 0 to $L-1$, where $L = 2^b$. Each n-th pixel is processed in an instant t_s, which represents the sampling time. Thus, one complete image can be processed at every m-th moment, where

$$m = M \times N \times t_s. \tag{2.2}$$

This equation must be changed if more than one pixel is processed per sample time.

The normalized histogram of each m-th image, $\mathbf{A}(m)$, is calculated and stored in the vector

$$\mathbf{p}(m) = [p_0(m), \ldots, p_k(m), \ldots, p_{L-1}(m)] \tag{2.3}$$

where each k-th component, $p_k(m)$, is defined as

$$p_k(m) = \frac{n_k(m)}{M \times N}, \tag{2.4}$$

in which $n_k(m)$ denotes the number of pixels with intensity k of the m-th image, described

as

$$n_k(m) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} c_{i,j}(n), \tag{2.5}$$

with $c_{i,j}(n)$ expressed as

$$c_{i,j}(n) = \begin{cases} 1 & \text{if } a_{i,j}(n) = k \\ 0 & \text{otherwise} \end{cases}. \tag{2.6}$$

Subsequently, after obtaining the normalized histogram, stored in the vector $\mathbf{p}(m)$, the Otsu algorithm calculates an optimal threshold between the two classes, i.e., C_0 and C_1. The optimal threshold called here as $k^*(m)$ can be characterized as

$$k^*(m) = \underset{0 \leq k \leq L-1}{\arg\max}\, \sigma_k^2(m). \tag{2.7}$$

where $\sigma_k^2(m)$ is the k-th between-class variance of the m-th image, defined as

$$\sigma_k^2(m) = \frac{(\mu_{L-1}(m) \times \omega_k(m) - \mu_k(m))^2}{\omega_k(m)\,(1 - \omega_k(m))}, \tag{2.8}$$

where $\omega_k(m)$ and $\mu_k(m)$ are the probability of class occurrence given a k threshold and the mean intensity value of the pixels up to the k threshold of the m-th image, respectively, meanwhile, $\mu_{L-1}(m)$ is the average intensity of the entire m-th image, called the global mean, with a value equal to $\mu_k(m)$ when $k = L-1$.

The variables $\omega_k(m)$ and $\mu_k(m)$ can be expressed as

$$\omega_k(m) = \sum_{i=0}^{k} p_i(m) \tag{2.9}$$

and

$$\mu_k(m) = \sum_{i=0}^{k} i \cdot p_i(m). \tag{2.10}$$

After finding the optimal threshold value, $k^*(m)$, the pixels of the input image, $\mathbf{A}(m)$, can be classified as a background or object (C_0 and C_1), generating a mask for the input image.

2.3 Hardware Proposal

In this section a fully parallel architecture of the Otsu method capable of processing images of any resolution is proposed, focused on obtaining high-speed processing. The details of the hardware implementation are described in the following subsections.

2.3.1 General Architecture

The general hardware architecture implemented is presented through a block diagram, shown in Figure 2.1. As can be observed, the architecture was developed according to the definition of Otsu's algorithm provided in Section 2.2. Therefore, it consists of five main modules: Normalized Histogram Module (NHM), Probability of Class Occurrence Module (PCOM), Mean Intensity Module (MIM), Between-Class Variance Module (BCVM), and Optimal Threshold Module (OTM).

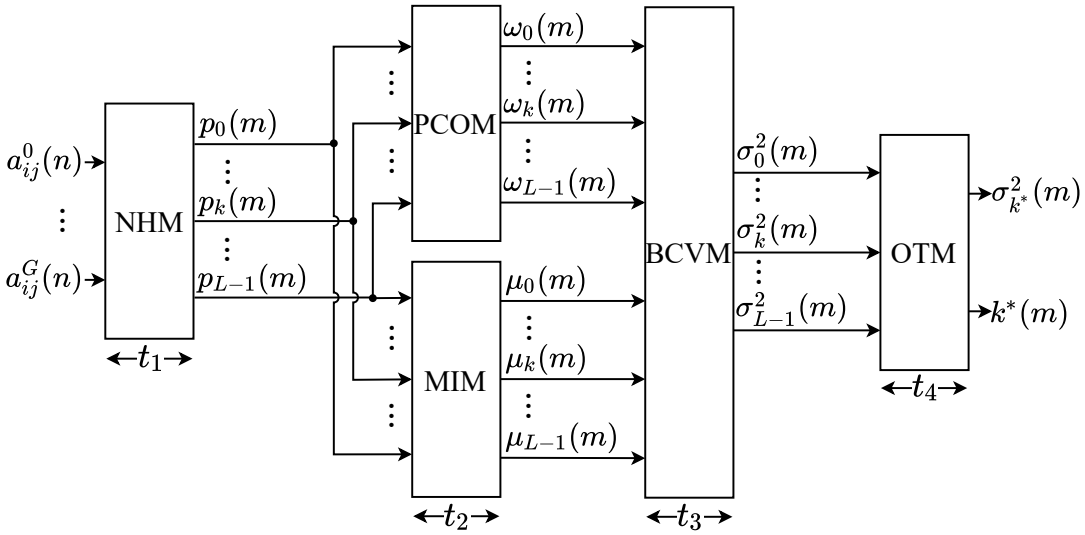

Figure 2.1: General architecture of the proposed Otsu Algorithm implementation.

Initially, the NHM module receives the parallel input of G image pixels, where G is the number of submodules internal to the NHM that simultaneously calculate the normalized histogram, according to Equations 2.3 and 2.4. Subsequently, the PCOM module uses the histogram components to calculate the class occurrence probabilities, according to the Equation 2.9, while the MIM module calculates the average intensities, based on Equation 2.10. The PCOM and MIM modules perform their calculations simultaneously. Afterward, these two modules outputs are supplied to BCVM, in which the values of the between-class variance are computed, according to the Equation 2.8. Finally, the calculated between-class variances are compared in the OTM to select the optimal threshold value, as described in Equation 2.7.

All variables and constants shown in Figure 2.1 were implemented in fixed-point to reduce the bit-width compared to floating-point implementations. The amount of bits used can be adjusted to adapt the precision of the results obtained to the desired application. Therefore, each pixel, $a_{ij}^g(n)$, of the input image were configured with 8 bits in the integer part (without sign), then $L = 2^8 = 256$ is defined. For the histogram components, $p_k(m)$, and the probabilities of class occurrence, $\omega_k(m)$, which has a positive value less than 1, only 24 bits are used in the decimal part. For the average intensity elements, $\mu_k(m)$, 8 bits are used in the integer part (without sign) and 24 bits in the decimal part. For the between-class variances, $\sigma_k^2(m)$, 27 bits are used in the integer part (one bit for sign) and

24 bits in the decimal part. Finally, for the optimal threshold, $k^*(m)$, only 8 bits are used in the integer part (without sign).

The modules of this architecture are pipelined, and the system operates on the same sample time, t_s. Nonetheless, each module has a different execution time, characterized here as t_1, t_2, t_3 and t_4. To minimize control, due to the lack of synchronism between the modules, the hardware proposed here defines $t_1 = t_2 = t_3 = t_4 = t_I$, where t_I is the time to process a complete image, being equal to the m-th moment that an image is processed. The time t_I is defined by the NHM block, since t_1 has the longest execution time. Thus, the system has an initial latency expressed as

$$D = 4 \times t_I \tag{2.11}$$

and a throughput characterized as

$$th = 1/t_I. \tag{2.12}$$

2.3.2 Normalized Histogram Module (NHM)

The NHM is responsible for generating the normalized histogram of the input image, by performing the equations 2.3 and 2.4. Usually, this step of the algorithm costs more clock cycles to complete than other steps, as the entire image needs to be scanned to obtain the normalized histogram components. We propose the parallelization of this step by calculating the components' partial values in a parallel way to optimize this process. Afterward, these values are summed to obtain their final values. The architecture of this module is shown in Figure 2.2.

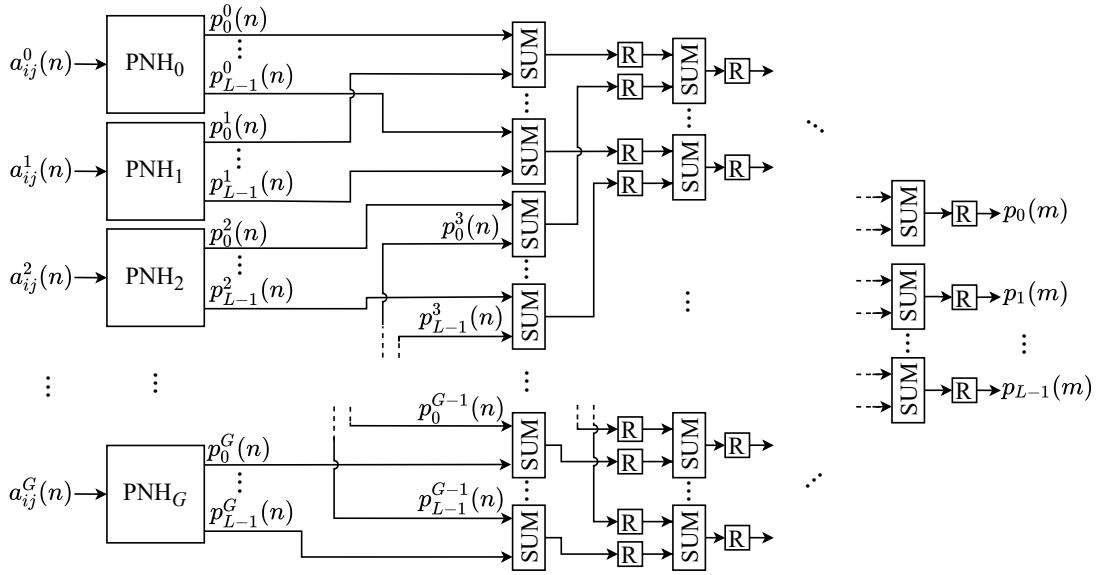

Figure 2.2: Architecture of the NHM.

As can be observed, the NHM module is constituted of G identical submodules, called Partial Normalized Histogram (PNH), responsible for computing the partial values of the

histogram components. Each g-th input pixel, $a_{ij}^{g}(n)$, is processed by the g-th PNH_g. Likewise, the PNH modules are internally constituted of L submodules, called Partial Component of the Normalized Histogram (PCNH), as shown in Figure 2.3. Thus, each k-th partial component of the histogram calculated by the g-th PNH, $p_k^g(n)$, is computed in parallel by a PCNH_k^g submodule. Figure 2.4 shows the internal circuit of each PCNH module, consisting of a comparator ($COMP_k^g$), an adder (SUM_k^g), two registers (R) and two constants (C_1 and C_2).

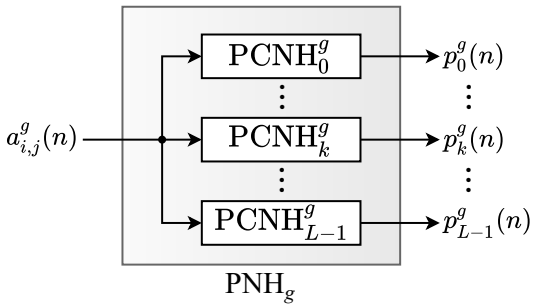

Figure 2.3: Architecture of the PNH_g.

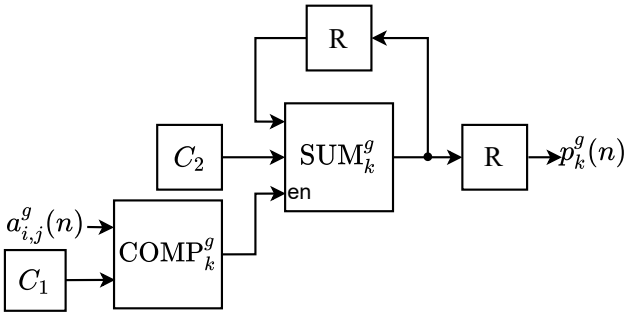

Figure 2.4: Architecture of the PCNH_k^g.

Initially, in each k-th comparator of the g-th PCNH_k^g, $COMP_k^g$, it is checked whether the input pixel, $a_{ij}^g(n)$, has value equal to C_1, according to Equation 2.6. The constant C_1 has a different value of k in each PCNH_k^g submodule. Following, the $COMP_k^g$ output, $c_{i,j}(n)$, which is a Boolean value, enables the adder, SUM_k^g, when equal to 1. Therefore, when SUM_k^g is enabled, the constant C_2 is summed with its previous value, thus operating as an accumulator. The constant C_2 is defined as $\frac{1}{M \times N}$, so it determines the value of $p_k(m)$ when summed n_k times, according to Equation 2.4. After entering all the image pixels, each PCNH_k^g outputs the k-th partial component of the normalized histogram computed in the g-th PNH, $p_k^g(n)$.

After that, the final value of each k-th component of the m-th image, $p_k(m)$, is obtained by summing all the k-th partial values provided as an output of each g-th PNH, $p_k^g(n)$. This sum is performed for each k-th component through an adder tree (SUM), as represented in Figure 2.2. This tree has a depth equal to $\log_2 G$. At the end, the value of the components of the normalized histogram, $p_k(m)$, is obtained, according to Equation 2.3.

Instead of processing 1 pixel per sample time in the histogram, the proposed architecture allows processing G pixels in parallel. Consequently, the amount of clock cycles required in this step is reduced and, thus, the processing time of a complete image, t_I. Consequently, the latency is also reduced, and throughput increased. With this scheme, based on the Equation 2.2, the value of t_I can be defined by

$$t_I = \left(\frac{M \times N}{G} + \lceil log_2(G) \rceil \right) \times t_s. \tag{2.13}$$

Through this equation, it is possible to observe that the higher the value of G, the lower the value of t_I and, consequently, better the performance achieved.

In each PCNH submodule, the constant C_1 assumes a grayscale value between 0 and $L-1$, and is represented with only 8 bits in the integer part (without sign). The constant C_2, which has a positive value less than 1, uses only 24 bits in the decimal part. This bit-width is also used for all the adders of the NHM module, as the normalized histogram components also assume positive values less than 1. All the k-th components of the histogram, $p_k(m)$, are transmitted in parallel to PCOM and MIM.

2.3.3 Mean Intensity Module (MIM)

The MIM calculates the average intensity value of the pixels up to level k, according to Equation 2.10. Each k-th average intensity, $\mu_k(m)$, is calculated in parallel. The architecture of this module, shown in Figure 2.5, consists of L gains submodules (G_k), $\frac{L}{2} \times \log_2 L$ adders (SUM) and one register (R) after each component.

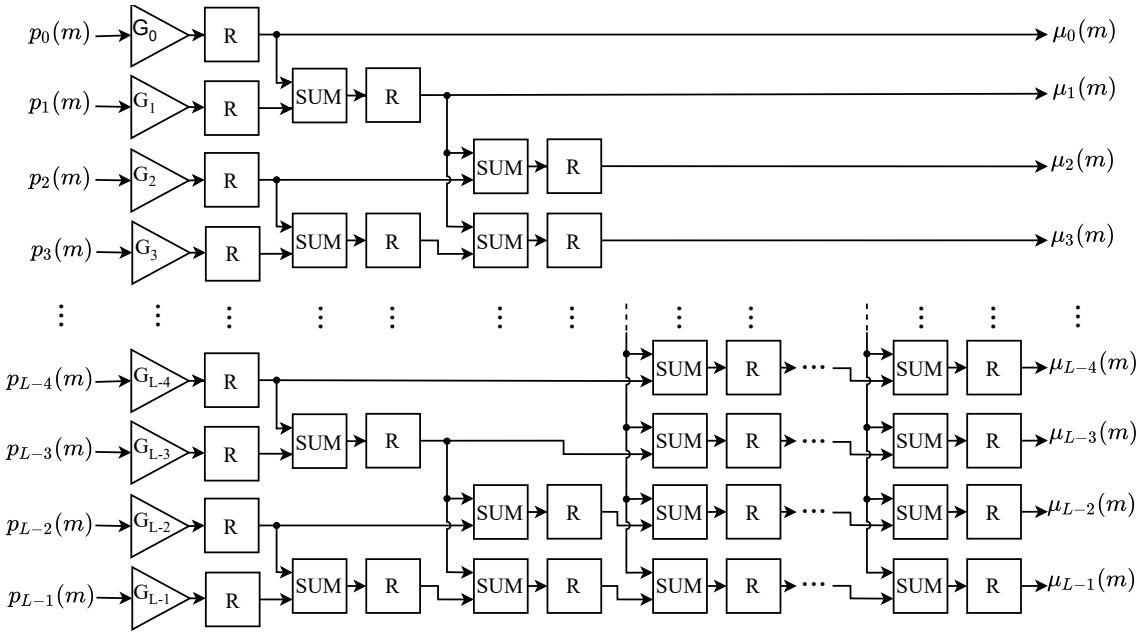

Figure 2.5: Architecture of the MIM.

Based on the Equation 2.10, each k-th $p_k(m)$ component of the normalized histogram

is first multiplied by a gain with the value of k. These gains are represented in the architecture block diagram by $G_0, ..., G_{L-1}$, and the index indicates the value of the applied gains. Thereafter, each k-th average intensity, $\mu_k(m)$, is obtained by summing the outputs of all gains with an index from 0 to k. The sum of these values is carried out in parallel based on the adder proposed by Ladner & Fischer (1980), with a maximum of $\log_2 L$ cascading adders using this technique.

All k-th gains of this module, G_k, have their output represented with 8 bit in the integer part (without sign) and 24 bits in the decimal part. Similarly, this bit resolution is also used for the adders, SUM, as the average intensity values of the pixels are at most equal to $L = 256$, for input images with pixels represented by 8 bits. All k-th average intensities, $\mu_k(m)$, are provided to BCVM in parallel.

2.3.4 Probability of Class Occurrence Module (PCOM)

The probability of class occurrence for a given threshold k, $\omega_k(m)$, is performed in PCOM based on Equation 2.9. This module has an architecture similar to MIM, but it does not have the gain submodule to weight the input. Therefore, the inputs are directly linked to the adders (SUM). Thus, this architecture is composed only of adders and registers, as shown in Figure 2.5.

According to Equation 2.9, the $\omega_k(m)$ values are obtained by adding all the components of the histogram from index 0 to k. Thus, using the parallel adder proposed by Ladner & Fischer (1980), all $\omega_k(m)$ values are computed simultaneously through the sum of the k-th entries $p_k(m)$.

The adders in this module were implemented for the same bit resolution of the inputs, $p_k(m)$, since the probability of class occurrence also assumes positive values less than 1. All k-th probabilities $\omega_k(m)$ calculated are propagated to the BCVM in parallel.

2.3.5 Between-Class Variance Module (BCVM)

The k-th between-class variance of the m-th image, $\sigma_k^2(m)$, are calculated by BCVM based on the Equation 2.8. The BCVM module is internally composed of L equal submodules, named Between-Class Variance of k (BCV_k), with the same architecture shown in Figure 2.6. Each k-th $\sigma_k^2(m)$ is computed in parallel by the k-th submodule BCV_k. This submodule consists of four multipliers ($MULT_{i,k}$), two subtractors ($SUB_{i,k}$), a point shift (BPC_K), a Look-Up Table (LUT_k) and eleven registers (R).

The Equation 2.8 is performed in parallel in this architecture. Therefore, the numerator and denominator are performed simultaneously. Concerning the numerator, it is obtained by first multiplying the k-th probability, $\omega_k(m)$, by the global mean, $\mu_{L-1}(m)$, on $MULT_{1,k}$. Subsequently, the k-th $SUB_{1,k}$ submodule perform the subtraction between $\mu_k(m)$ and the $MULT_{1,k}$ output. Finally, the result of this subtraction is multiplied by itself in the k-th $MULT_{2,k}$. Regarding the denominator, it is calculated by initially subtracting the k-th probability, $\omega_k(m)$, from the value 1 in the k-th $SUB_{2,K}$. Lastly, the result is multiplied by the same $\omega_k(m)$ in the k-th $MULT_{3,k}$.

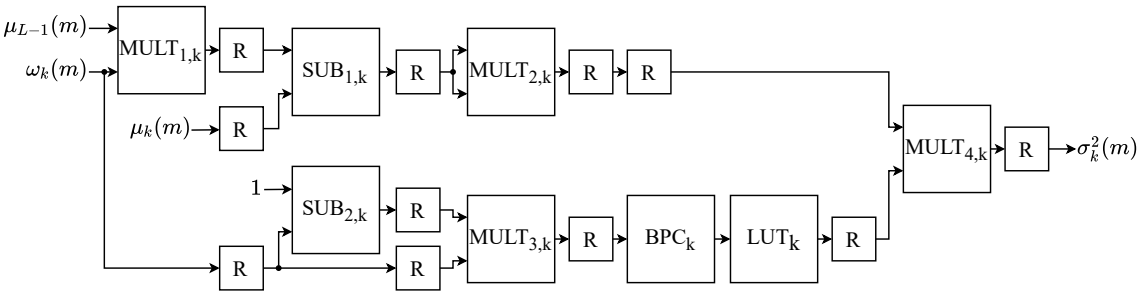

Figure 2.6: Architecture of the BCV_k submodule.

The arithmetic operation of division is highly costly to the hardware in terms of processing speed, being the architecture's bottleneck due to the highest critical time. One way to avoid using the division is to multiply the numerator by the reciprocal of the denominator. By definition, the reciprocal of a number is its inverse. Thereupon, the denominator's reciprocal can be approximated for a range of predefined values and stored in a LUT. Thus, the division can be performed using only one LUT and a multiplier, LUT_k and $MULT_{4,k}$, consequently increasing the throughput of the implementation.

Thus, each k-th value in the output of $MULT_{3,k}$ has a reciprocal approximated value in the k-th LUT_k. This LUT was configured with a depth of L, storing words of 33 bits, where 9 bits represent the integer part (one bit for sign) and 24 bits the fractional. The mapping of the output value of each k-th $MULT_{3,k}$ to an address of the LUT_k is performed by shifting the binary point eight bits to the right by the k-th BPC_k. The approximate value of the reciprocal shown at the output of each k-th LUT_k is multiplied by the calculated value of the numerator in $MULT_{4,k}$. The result of this multiplication is the k-th between-class variance of the m-th image, $\sigma_k^2(m)$.

The subtractor $SUB_{1,k}$ was configured with 9 bits in the integer part (one bit for sign) and 24 bits in the decimal part, while $SUB_{2,k}$, uses only 24 bits in the decimal part. The multiplier $MULT_{1,k}$ uses 8 bits in the integer part (without sign) and 24 bits in the decimal part, while $MULT_{2,k}$ uses 18 bits in the integer part (one bit for sign) and 24 bits in the decimal part. Meanwhile, $MULT_{3,k}$ was configured with only 24 bits in the decimal part, and $MULT_{4,k}$ uses 27 bits in the integer part (one bit for sign) and 24 bits in the decimal part. Each k-th between-class variance calculated is propagated in parallel to the OTM.

2.3.6 Optimal Threshold Module (OTM)

The OTM module performs the last step of the Otsu algorithm, responsible for comparing all k-th values of the between-class variance, $\sigma_k^2(m)$, to determine the optimal threshold of the m-th image, $k^*(m)$, based on Equation 2.7. The architecture of this module is shown in Figure 2.7. As can be observed, it consists of $L-1$ comparators ($COMP_k$), $2\cdot(L-1)$ multiplexers (MUX) and a register (R) after each component.

According to Equation 2.7, the optimal threshold of the m-th image, $k^*(m)$, is the threshold value for which the highest value of between-class variance is obtained. For this purpose, the between-class variances of the m-th image, $\sigma_k^2(m)$, are compared through a comparator tree. Each k-th $COMP_k$, compares whether a given variance $\sigma_k^2(m)$ is greater

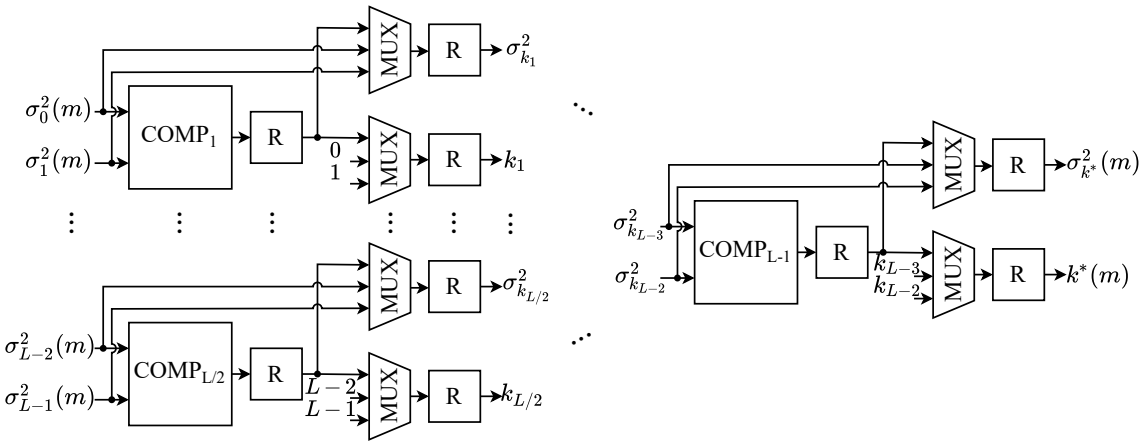

Figure 2.7: Architecture of the OTM.

than the other, $\sigma^2_{k+1}(m)$. This comparator is used as a key selector of two multiplexers, defining on the outputs the largest variance value that was compared and its respective threshold k. All outputs of the multiplexers are passed to the next branch of the tree until the last comparator, $COMP_{L-1}$, in which the optimal threshold value of the m-th image, $k^*(m)$, should be selected as the output of the multiplexers, as well as its between-class variance, $\sigma^2_{k^*}(m)$.

Therefore, the optimal threshold of the m-th input image is determined by the proposed architecture of a fully parallel design. A new value of $k^*(m)$ is computed for every m-th instant.

2.4 Results

The architecture presented in the previous section was developed on an FPGA Arria 10 GX 1150, and the analysis of the synthesis results was carried out concerning hardware area occupation, throughput, and power consumption.

2.4.1 Hardware Area Occupation Analysis

Initially, the hardware area occupation analysis was performed for the architecture with one PNH module only. The results are shown in Table 2.1. The first to the third columns indicate the number of logical cells occupied (n_{LC}), the number of multipliers implemented using DSP blocks (n_{Mult}), and the number of block memory bits (n_{BitsM}), respectively. It also presents the resources used in percentage.

Table 2.1: Hardware area occupation for $n_{PNH} = 1$.

n_{LC}	n_{Mult}	n_{BitsM}
591,946 (69.3%)	1,222 (80.5%)	2,162 K (3.9%)

As can be observed, only 4% of the block memory bits available have been used, while 69% of the logic cells were occupied. The most used resource was the multipliers, occupying 80% of the total DSPs available. Therefore, the data presented in Table 2.1 demonstrate the feasibility of implementing the proposed architecture in the target FPGA. Besides, the Arria-10 FPGA still has resources available that can be used to implement additional logic, thus, allowing to increase the number of PNH modules.

PNH modules require only logical cells for their implementation since they are not designed with multipliers and memories. Therefore, the 30% of unused logic cells in the Arria-10 allows increasing the number of PNH modules. Hence, we also analyzed the area occupation for the Arria-10 FPGA by increasing the number of PNH modules (n_{PNH}). It is presented in Table 2.2, the number of occupied logical cells as there is no change in the use of other resources.

Table 2.2: Number of logic cells required per PNH module.

n_{PNH}	n_{LC}
1	591,946 (69.3%)
2	629,096 (73.6%)
4	680,884 (79.7%)
6	740,368 (86.6%)
8	788,092 (92.2%)
10	848,630 (99.3%)

Figure 2.8 shows the curve obtained by linear regression using the set of values presented in Table 2.2. The equation associated with the regression analysis is expressed by

$$n_{LC} = \left\lceil (2.8 \cdot n_{PNH} + 56.9) \cdot 10^4 \right\rceil . \tag{2.14}$$

This equation can obtain the number of logical cells occupied by the architecture without the NHM module when defining $n_{PNH} = 0$. Through this, it is possible to observe that increasing the amount of PNH modules causes a minor impact on the total number of occupied hardware resources. This characteristic is one of the advantages of the proposed architecture, which allows increasing the degree of parallelism of the implementation with a few more hardware resources.

Therefore, Equation 2.14 allows estimating the maximum number of PNH modules an FPGA can support. It is presented in Table 2.3, the maximum number of PNH modules supported on different commercial FPGAs (Intel 2020*b*, Intel 2020*a*). The first and second columns present the label and FPGA, respectively, while the third and fourth columns present the number of logical cells available (n_{LC}^{max}) and the maximum number of PNH modules that can be implemented with these resources (n_{PNH}^{max}). Hence, a high degree of parallelism can be achieved through our proposed architecture, limited only by the FPGA resources available.

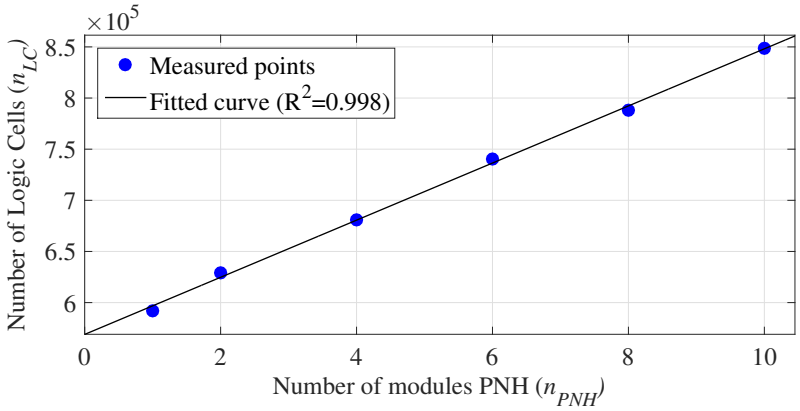

Figure 2.8: Linear regression curve for the hardware area occupation (logic cells) per n_{PNH}.

Table 2.3: Estimated number of PNH modules, n_{PNH}^{max}, for some commercial FPGAs.

Label	**FPGA**	$\boldsymbol{n_{LC}^{max}}$	$\boldsymbol{n_{PNH}^{max}}$
$FPGA_1$	Arria 10 GX 1150	854,400	10
$FPGA_2$	Agilex AGF 027	1,825,600	44
$FPGA_3$	Stratix 10 GX 10M	6,932,160	227

2.4.2 Time Processing Analysis

The data related to the system's processing time was obtained considering a clock cycle of 13.3ns, which is defined by its critical path. Moreover, as the circuit operates with the same clock, the sampling time is also defined as $t_s = 13.3$ns.

The system's processing time, for different amounts of n_{PNH}, is presented in Table 2.4. The first column indicates the number of n_{PNH}. Meanwhile, from the second to fourth columns are shown, respectively, the image processing time, t_I, defined according to Equation 2.13, the initial system latency, D, according to Equation 2.11, and the throughput, th, which in this proposal consists in the number of images processed per second (IPS), determined through the Equation 2.12. According to Equation 2.13, the processing time of an image depends on its size. Thus, the data presented in Table 2.4 concern the processing of a $3,840 \times 2,160$ image with 4K resolution.

As can be observed in Table 2.4, the value of n_{PNH} is directly proportional to th and inversely proportional to t_I and D. Thus, the more PNH modules employed in the implementation, the better the system performance. Figure 2.9 shows the curve obtained by linear regression that relates the values of n_{PNH} and th. The equation associated with this curve is expressed by

$$th = th(n_{PNH} = 1) \times n_{PNH}. \tag{2.15}$$

Table 2.4: System's processing time.

n_{PNH}	t_I (ms)	D (ms)	th (IPS)
1	110.32	441.28	9
2	55.16	220.64	18
4	27.58	110.32	36
6	18.39	73.56	54
8	13.79	55.16	72
10	11.03	44.12	90

According to 2.12, and 2.13, the Equation 2.15 can be rewriten as

$$th = \frac{1}{\left(\frac{M \times N}{1} + \lceil log_2(1) \rceil\right) \times t_s} \times n_{PNH} = \frac{n_{PNH}}{M \times N \times t_s}. \tag{2.16}$$

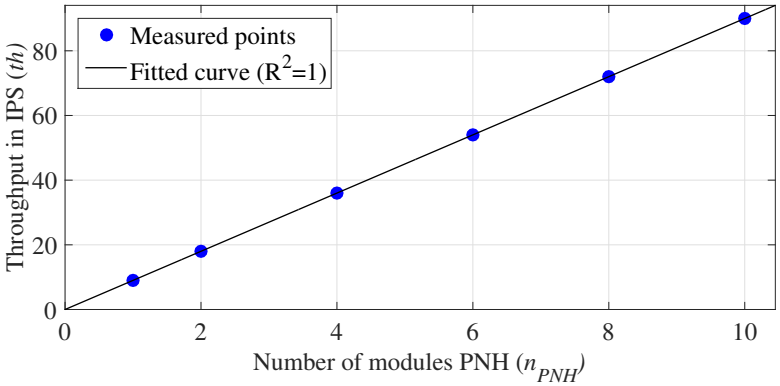

Figure 2.9: Throughput in IPS, th, per n_{PNH}.

Equation 2.16, allows to define t_I values as

$$t_I = \frac{1}{th} = \frac{M \times N \times ts}{n_{PNH}}. \tag{2.17}$$

When comparing this equation to Equation 2.13, it is observed that the only difference between them is the absence of the sum of the logarithm. The reason for that is

$$\lceil log_2(n_{PNH}) \rceil \ll \frac{M \times N}{n_{PNH}}. \tag{2.18}$$

Thus, this component can be disregarded in the calculation of t_I.

Afterward, the proposed architecture's processing time was analyzed for other commercial FPGAs, previously presented in Table 2.3, adopting for each FPGA an $n_{PNH} = n_{PNH}^{max}$. For this purpose, the values t_I and th were defined according to Equations 2.16 and 2.17, respectively, and different image resolutions considered. The results are shown in

Table 2.5.

Table 2.5: Values of t_I and th for some commercial FPGAs for different image sizes.

Image Size	$\mathbf{FPGA_1}$		$\mathbf{FPGA_2}$		$\mathbf{FPGA_3}$	
	t_I (ms)	th (IPS)	t_I (ms)	th (IPS)	t_I (ms)	th (IPS)
4K ($3,840 \times 2,160$)	11.03	90	2.51	398	0.49	2,057
8K ($7,680 \times 4,320$)	44.13	22	10.03	99	1.94	514
10K ($10,240 \times 4,320$)	58.80	16	13.40	74	2.60	385
16K ($15,360 \times 8,640$)	176.50	5	40.11	24	7.78	128

All FPGAs models analyzed achieved a high throughput in processing images with 4K resolution, allowing real-time processing of 4K videos in all models. For images with 8K and 10K resolutions, real-time processing proved to be more viable in the $FPGA_2$ and $FPGA_3$. Finally, a high throughput was also achieved by the $FPGA_3$ in processing 16K images, allowing real-time processing of videos with this resolution. Therefore, the $FPGA_3$ offers better performance due to the high number of PNH modules that can be implemented, i.e., the increased architecture's parallelism.

2.5 Comparison with State-of-the-Art Works

Comparisons with the state-of-the-art works were performed for the three commercial FPGAs previously mentioned. The architectures were implemented using the maximum number of PNH modules, presented in Table 2.3. The results were compared only with works that presented data about hardware area occupation and processing time.

2.5.1 Hardware Occupation Comparison

Table 2.6 presents the comparison of the hardware area occupation. The first column indicates the reference analyzed. The second to fifth columns show the FPGA, the number of logical cells, multipliers, and the number of bits of memory blocks, respectively, of the reference. Meanwhile, the last four columns present the commercial FPGA to be compared and the ratio between the hardware components used in our proposed implementation, n_{work}, and those used in the literature, n_{ref}. The ratio of the hardware occupation can be expressed as

$$R_{occupation} = \frac{n_{work}}{n_{ref}}, \tag{2.19}$$

where n_{work} and n_{ref} can be replaced by n_{LC}, n_{MULT}, or n_{BitsM}. This ratio was calculated using the hardware occupancy data of the proposed architecture previously presented, using the values of n_{MULT} and n_{BitsM} provided by Table 2.1 and n_{LC} given by Table 2.3.

Table 2.6: Hardware occupation comparison with related works.

Ref.	**FPGA$_{ref}$**	n_{LC}	n_{Mult}	n_{BitsM}	**FPGA$_{work}$**	**R$_{occupation}$** n_{LC}	n_{Mult}	n_{BitsM}
Das et al. (2014)	Spartan 6	2,975	-	792K	FPGA$_1$	$\approx 287.19\times$	-	$\approx 2.73\times$
					FPGA$_2$	$\approx 613.65\times$	-	$\approx 2.73\times$
					FPGA$_3$	$\approx 2,330.13\times$	-	$\approx 2.73\times$
Torres-Monsalve & Velasco-Medina (2016)	Cyclone IV	2,440	46	10.94K	FPGA$_1$	$\approx 350.16\times$	$\approx 26.56\times$	$\approx 197.62\times$
					FPGA$_2$	$\approx 748.20\times$	$\approx 26.56\times$	$\approx 197.62\times$
					FPGA$_3$	$\approx 2,841.05\times$	$\approx 26.56\times$	$\approx 197.62\times$
Pandey & Karmakar (2019)	Virtex 5	1,031	5	144K	FPGA$_1$	$\approx 828.71\times$	$\approx 244.40\times$	$\approx 15.01\times$
					FPGA$_2$	$\approx 1,770.71\times$	$\approx 244.40\times$	$\approx 15.01\times$
					FPGA$_3$	$\approx 6,723.72\times$	$\approx 244.40\times$	$\approx 15.01\times$

The proposal presented by Das et al. (2014) was deployed on a Spartan-6 LX45 FPGA and occupied an area of 1,859 LUTs and 44 RAM blocks. This FPGA uses about 1.6 logic cells (LC) per LUT, having used about 2,975 LC, and each block of RAM has 18 Kbits, so 792 Kbits of memory is used (Xilinx 2010). The number of multipliers used was not available.

In Torres-Monsalve & Velasco-Medina (2016), results were presented for two different implementations of the Otsu method, deployed on the Altera Cyclone IV EP4CE115-F29C6N FPGA. The comparison with this work was performed considering the best results presented by these authors, i.e., the method using logarithmic functions. The implemented hardware used 2,440 LC, 10.94 Kbits of memory, and 46 multipliers.

The proposal presented in Pandey & Karmakar (2019) uses the Virtex-5 xc5vfx70tffg1-136-1 FPGA and occupied a total of 161 slices, 4 BRAMs and 5 multipliers. Each slice of this FPGA has 4 LUTs of 6-input, hence, 6.4 LC per slice was used, and each memory block has 36 Kbits (Xilinx 2006). Thus, about 1,030 LC and 144 Kbits of memory were used.

Through Table 2.6, we found that in all comparisons performed our design presented a more significant hardware area occupation, due to the high degree of parallelism adopted in our proposed implementation, which results in more hardware resources.

2.5.2 Time Processing Comparison

Table 2.7 presents a throughput comparison. The analysis was carried out for all the commercial FPGAs previously presented. As can be seen, the first column presents the analyzed reference, while the second to fourth columns show the image resolution (IR), the clock (Clk), and the throughput achieved (th_{ref}), respectively, by the reference. The last three columns indicate, respectively, the FPGAs analyzed, the throughput (th_{work}), and speedup reached with our architecture. The values of th_{work} are calculated according to the Equation 2.16, employing the same clock and image size adopted by the compared reference. The speedup calculation is defined as

$$Speedup = \frac{th_{work}}{th_{ref}}. \tag{2.20}$$

Table 2.7: Throughput comparison with related works.

Ref.	**IR**	**Clk (ns)**	th_{ref} **(IPS)**	**FPGA**	th_{work} **(IPS)**	**Speedup**
Das et al. (2014)	280×265	10.00	671.59	$FPGA_1$	$13,469.83$	$\approx 20.06\times$
				$FPGA_2$	$59,088.95$	$\approx 87.98\times$
				$FPGA_3$	$298,621.34$	$\approx 444.65\times$
Torres-Monsalve & Velasco-Medina (2016)	$1,280 \times 1,024$	8.72	43.72	$FPGA_1$	874.90	$\approx 20.01\times$
				$FPGA_2$	$3,848.92$	$\approx 88.03\times$
				$FPGA_3$	$19,833.44$	$\approx 453.65\times$
Pandey & Karmakar (2019)	640×480	39.72	16.39	$FPGA_1$	819.43	$\approx 49.99\times$
				$FPGA_2$	$3,602.87$	$\approx 219.82\times$
				$FPGA_3$	$18,494.20$	$\approx 1,128.38\times$

In Das et al. (2014), the runtime results are presented using a clock of 10ns and a 280×265 input image, with pixels represented by 8 bits. The processing time is 1.489ms, achieving a throughput of 671.59 IPS.

In Torres-Monsalve & Velasco-Medina (2016), time and latency data were presented considering a clock of 8.72ns and the processing of a $1,280 \times 1,024$ image, also with pixel representation for 8 bits. As the processing time of an image was not presented by the authors, it was estimated as the clock value times the number of cycles required to enter an image and obtain its respective output. The processed image's input and output are performed serially, requiring $1,280 \times 1,024$ clocks cycles to scan the entire image. The latency indicated for the implementation of the Otsu method using logarithmic optimization is equal to 536. Thus, the calculated time for processing an image is equal to $(1,280 \times 1,024 \times 2 + 536) \times 8.72 \cdot 10^{-9} \approx 22,87$ms per image. Hence, achieving a throughput of 43.72 IPS.

The proposal presented by Pandey & Karmakar (2019) displays results from throughput for processing a 640×480 image, with pixels represented by 8 bits, and adopting a clock of 39.72ns. The throughput shown indicates the number of megabits processed per second (Mbps), which in turn is 40.28Mbps. As each pixel has 8 bits, the number of images processed per second can be obtained by $\frac{40.28 \cdot 10^6}{8 \times 640 \times 480} \approx 16,39$IPS.

According to the results presented in Table 2.7, our proposed architecture obtained better throughput values than other works in the literature in all comparisons performed. High speedup values were obtained, varying between $20\times$ and $1,128\times$. The performance achieved by the developed architecture is mainly due to the high degree of parallelism explored in this proposal, with a focus on parallelizing the calculation of the histogram. In contrast, works in the literature present implementations with a low degree of parallelism, and are focused on optimizing the calculation of between-class variance.

2.5.3 Power Consumption Comparison

Table 2.8 presents a comparison of our design's dynamic power consumption compared to other proposals in the literature. The dynamic power consumption can be expressed as

$$P_d \propto N_g \times F_{clk} \times V_{DD}^2 \tag{2.21}$$

where P_d is the dynamic power consumption, N_g is the number of hardware components, F_{clk} is the frequency, and V_{DD} is the supply voltage. The frequency at which a circuit can operate is proportional to the voltage (McCool et al. 2012, Silva et al. 2020). Thus, the Equation 2.21 can be rewritten as

$$P_d \propto N_g \times F_{clk}^3. \tag{2.22}$$

For all comparisons, the value of N_g was calculated as

$$N_g = n_{LC} + n_{Mult}. \tag{2.23}$$

Based on the Equation 2.22, the saved dynamic energy, S_d, can be expressed by

$$S_d = \frac{N_g^{ref} \times (F_{clk}^{ref})^3}{N_g^{work} \times (F_{clk}^{work})^3} \tag{2.24}$$

where N_g^{ref} and F_{clk}^{ref} are the number of hardware components and clock of literature works, respectively, and N_g^{work} and F_{clk}^{work} are the number of components and the clock adopted in this proposal to obtain the same throughput of the reference to which it is being compared.

Table 2.8: Dynamic power comparison with related works.

Ref.	N_g^{ref}	F_{clk}^{ref} **(MHz)**	N_g^{work}	**FPGA**	F_{clk}^{work} **(MHz)**	S_d
Das et al. (2014)	2,975	100.00	593,168	$FPGA_1$	4.98	$\approx 2.82 \cdot 10^1\times$
				$FPGA_2$	1.13	$\approx 1.13 \cdot 10^3\times$
				$FPGA_3$	0.22	$\approx 4.03 \cdot 10^4\times$
Torres-Monsalve & Velasco-Medina (2016)	2,486	114.62	593,168	$FPGA_1$	5.73	$\approx 2.33 \cdot 10^1\times$
				$FPGA_2$	1.30	$\approx 9.33 \cdot 10^2\times$
				$FPGA_3$	0.25	$\approx 3.46 \cdot 10^4\times$
Pandey & Karmakar (2019)	1,036	25.175	593,168	$FPGA_1$	0.50	$\approx 1.55 \cdot 10^2\times$
				$FPGA_2$	0.11	$\approx 6.80 \cdot 10^3\times$
				$FPGA_3$	0.02	$\approx 9.98 \cdot 10^5\times$

Table 2.8 shows that our implementation presents a significant reduction in energy consumption. The results presented indicate an energy-saving, regarding works in the literature, between $2.33 \cdot 10^1$ and $9.98 \cdot 10^5\times$. This reduction is obtained through the high degree of parallelism of the technique, which allows obtaining high throughput with the circuit operating at a low clock frequency. Therefore, although the circuit uses many hardware components, the energy consumed is reduced due to the low-frequency operation.

2.6 Conclusions

This chapter presented a parallel implementation proposal of the Otsu method in FPGA. The hardware architecture was developed using RTL design and fixed-point representation. All the implementation details were presented, and the synthesis results related to the hardware area occupation and processing time. The results showed that the proposed architecture achieved high throughput, enabling real-time processing of high-resolution videos. Comparisons with state-of-the-art works were also performed regarding the hardware area occupancy, throughput, and dynamic power consumption. Our proposed architecture outperformed the compared ones in terms of throughput and power consumption, achieving a speedup from $20\times$ until $10^3\times$ and reducing the power from $23\times$ until $10^5\times$. However, the amount of hardware resources required is higher than in other architectures due to the high degree of parallelism adopted in our method.

Chapter 3

High-Throughput Worm Segmentation

This chapter proposes the segmentation of worm images using the architecture described in Chapter 2. Results regarding resource utilization, throughput, and validation of the segmentation in hardware are provided. These results are used to evaluate whether the hardware architecture meets the real-time requirements of the system.

3.1 Introduction

In recent years, several automated tracking systems have been proposed for the behavioral analysis of *C. elegans*, which is a species of nematode worm widely used in biology research because of its small and well-characterized nervous system. These systems can be divided into two general categories, depending on the number of worms tracked (McDiarmid et al. 2018, Brown & Schafer 2015). Some works in the literature propose tracking a single worm to obtain high-resolution images that provide more details for accurate quantification of the animal's behavior (Hebert et al. 2021, Moy et al. 2016, Kwon et al. 2013, Yemini 2013, Faumont et al. 2011, Leifer et al. 2011, Wang et al. 2009, Tsibidis & Tavernarakis 2007, Cronin et al. 2006, Feng et al. 2004, Baek et al. 2002). These systems usually require motorized stages to keep the worm centered in the camera's field of view. Meanwhile, other works propose tracking multiple worms by capturing many worms in a single field of view. Such systems reduce the amount of detail in the representation of each worm, but increase throughput by allowing the tracking of multiple worms simultaneously (Layana Castro et al. 2021, Koopman et al. 2020, Bornhorst et al. 2019, Perni et al. 2018, Risse et al. 2017, Roussel et al. 2015, Moore et al. 2013, Mathew et al. 2012, Swierczek et al. 2011, Ramot et al. 2008, Tsechpenakis et al. 2008, Dusenbery 1985).

Because of the expensiveness and reduced throughput of single-worm tracking systems, the use of high-resolution cameras has been explored in multi-worm tracking systems to allow the capture of each worm with a high degree of detail. In Barlow et al. (2021), a multi-worm tracking system is proposed, with an array of six 12-megapixel cameras being used to simultaneously capture the image of all the wells of a 96-well plate with a resolution of 80 pixels/mm at 25 frames per second (FPS). This system enables the analysis of multiple worms simultaneously, representing each worm with sufficient resolution to extract accurate features. However, the experiments performed with this system

generate a massive volume of data, with the camera array producing about 6.5 Terabytes per hour (TB/h). The high memory requirement and, subsequently, the long processing time for this data are considered the main disadvantages of using such systems. Thus, the aim is to reduce this problem by processing the data in real-time.

A multi-worm tracking system with real-time processing is proposed by Swierczek et al. (2011), in which high-resolution cameras (4 Megapixels, 30 FPS, 41.2 pixels/mm) are used to track many worms on a single plate. In this paper, two different software are proposed, one for real-time image analysis, Multi-Worm Tracker (MWT), and another for offline analysis of several features, Choreography. The real-time processing provides only low-level features, such as the position and outline of the worm. Because MWT runs on CPU, system delays may occur due to a heavy workload or operating system call, in which case the software may skip some frames and process only the most recent one. Thus, the system does not work strictly in real-time.

In White et al. (2013) a high-throughput tracking system is also proposed to identify the stage of worm development. The software developed, called DevStaR, uses computer vision and machine learning techniques to segment images and count animals belonging to different developmental stages in a mixed population. This software processes images with a resolution of 1200×1600 pixels in an average time of 15 seconds. To carry out the analysis of a 96-well plate takes about 1.5 minutes. The data presented in this paper were obtained by running the software on a 16-core Linux box with 24 GB RAM.

The multi-worm tracking systems presented in the literature can achieve high throughputs, allowing them to perform even some simple analysis in real-time. However, it is still necessary to achieve higher throughputs that permit the entire system to operate in real-time. A possible way to get better performance results is through the development of dedicated hardware.

Thus, this chapter proposes to apply the Otsu method implemented in FPGA in the segmentation of worms, using the architecture presented in Chapter 2. In this chapter, results of hardware occupation and processing time are analyzed using different numbers of bits in the decimal part of the architecture and the representation of the image pixels, seeking to understand the impacts of this on throughput and hardware resource utilization. In addition, images of the worms captured by the system presented in Barlow et al. (2021) are utilized to validate the segmentation in hardware. Through the analysis of these data, it is possible to understand better the gains achievable with the development of the entire system in hardware.

The remainder of this chapter is organized as follows: Section 3.2 presents the tracking system used to capture worm images; Section 3.3 shows hardware performance evaluation metrics; while Section 3.4 provides the results related to hardware occupation, throughput, and segmentation validation. Finally, Section 3.5 presents the final considerations about this chapter.

3.2 Tracking System

This section presents the structure used in the PHENOSPACE project for tracking worms in multiwell plates. Details about the imaging system, the amount of data produced

by recording the experiments, and the image processing are provided.

3.2.1 Image Acquisition Framework

The PHENOSPACE project conducts research on behavioral genomics of the nematode *C. elegans*, exploring molecular mechanisms underlying nervous system function to identify genes and gene networks that affect behavior. This project is associated with the Behavioral Genomics Lab, which uses the high-resolution camera system proposed in Barlow et al. (2021) to track worms in multiwell plates. The imaging system presented in Barlow uses six 12 Megapixel cameras arranged as a 3×2 array. Through them, images of 96-well plates are captured simultaneously, with a resolution of 80 pixels/mm at a rate of 25 FPS. A public dataset with videos captured by this framework was made available by McDermott-Rouse* et al. (2021*b*), and the recorded experiments are described at McDermott-Rouse et al. (2021*a*). The videos provided in this dataset have a resolution of 3036×3036 pixels.

In the Behavioral Genomics Lab, five identical structures were built with this imaging system, seeking to increase the system throughput by performing parallel experiments. Thus, there are a total of 30 cameras recording videos from 480 wells simultaneously. The five imaging systems mounted in the Behavioral Genomics Lab are shown in 3.1a. The details of each structure with the camera arrays and the multiwell plates are presented in 3.1b. Figure 3.2 shows the image of the 480 wells (A), with the approximation of a set of wells (B) and a single well (C), highlighting one of the worms present in it to demonstrate the degree of details of the worms in the images obtained with this structure (D).

(a) (b)

Figure 3.1: (a) Structures with camera array for recording the worms in the Behavioral Genomics Lab. (b) Arrangement of the camera array and plates for recording the worms.

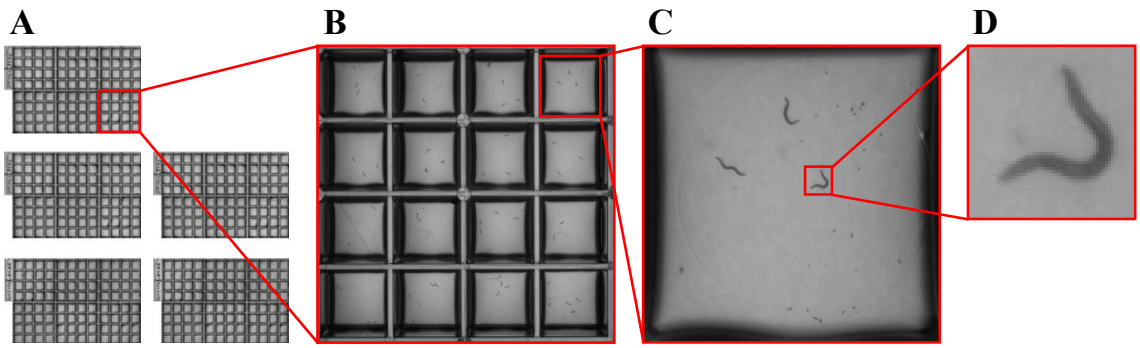

Figure 3.2: Images of the plates with the worms captured by the camera arrays of all five structures.

As described in Barlow et al. (2021), video recording using the proposed camera array generates approximately 6.5 TB/h. When the five available structures perform experiments simultaneously, this value increases to 32.5 TB/h. Thus, this system generates a massive volume of data that requires a lot of storage space and a long time to process. Therefore, it is desired to develop an implementation of high-throughput capable of processing the videos in real-time. Through this solution, all behavioral analyzes must be performed in real-time, eliminating the need to store videos. In addition, this processing speed will also allow the development of more complex studies on behavioral genomics.

3.2.2 Image Processing

Currently, the videos taken by this framework are processed offline by the Tierpsy Tracker software (Barlow et al. 2021, Javer et al. 2018). Even using GPUs, the implementation cannot perform real-time processing. One way to meet the timing requirements of the system is to develop dedicated hardware for the worm tracking algorithms. Through hardware implementation, it is possible to exploit the inherent parallelism of images. In the specific case of the videos captured by the Behavioral Genomics Lab structure, each well and, in some instances, each worm can be processed in parallel to improve system performance. With the development in FPGA, it is also possible to have throughput gains by employing multiple FPGAs, allowing further parallelization of the process.

One of the first steps in worm tracking is image segmentation, in which pixels are classified as belonging to the worm region or background. In the Tierpsy Tracker software, the Otsu method is one of the techniques employed in this process. Therefore, this chapter will discuss the application of the hardware architecture presented in Chapter 2 in the real-time segmentation of worms. The validation of the segmentation results obtained by the hardware architecture will be performed using images of the worms extracted from the videos available in the plubic dataset (McDermott-Rouse* et al. 2021*b*).

3.3 Image Segmentation with Designed Hardware

This section indicates the metrics adopted to evaluate the worm segmentation using the hardware proposed in Chapter 2, specifying the architecture configuration for which the results were obtained.

3.3.1 Hardware Setup

In Chapter 2, synthesis results related to hardware occupancy and processing time of the proposed architecture were presented for different FPGA models. The data presented for each model considered the implementation of the maximum number of modules PNH supported. It was also considered the utilization of 24 bits in the decimal part of the architecture and 8 bits in image pixel representation.

This chapter presents results for resource utilization, processing time, and validation of the worm image segmentation. These data are obtained considering the synthesis of the hardware architecture employing ten PNH modules. The results are analyzed with the reduction of the number of bits used in the decimal part of the architecture and the representation of the image's pixels, seeking to obtain higher throughputs. For this analysis, the decimal part of the architecture assumes values of 24, 19, and 14 bits, while the image pixels are represented by 8, 7, and 6 bits. The reduction of the number of bits used in the representation of the image pixels occurs by discarding the least significant bits that represent the intensity value. Only the NHM module does not have the number of bits of the decimal part of its components changed, being adopted a 32-bit resolution, because of the higher error accumulation in this stage.

3.3.2 Evaluation Metrics

Hardware Area Occupation

The hardware area occupation is analyzed regarding the number of logic cells occupied, multipliers implemented in DSP, and memory bits used. All the results presented were acquired by synthesizing the architecture for the Arria 10 GX 1150 FPGA.

Time Processing

The results related to the system processing time considering the processing of an image with dimensions 3036×3036, resolution equal to that of the videos available in the public dataset (McDermott-Rouse* et al. 2021*b*). The processing time of an image was calculated using Equation 2.17. Meanwhile, the initial system latency and throughput were calculated, respectively, through Equations 2.11 and 2.12.

Segmentation Validation

The validation of the worm segmentation utilizing the designed hardware was performed using a video with 6 minutes of duration, captured at the rate of 25 FPS, available

in the public dataset (McDermott-Rouse* et al. 2021*b*). From this video, 1800 frames were extracted with a spacing of 5 frames between them. The selected frames were cropped to obtain images with a resolution of 240×320, covering a region containing three worms. The evaluations carried out considered as golden standard of the images the segmentation results obtained through the Otsu method implemented in software.

The employed metrics use the golden standard to evaluate the classification of each pixel of the images segmented by the hardware as true positive (TP), false positive (FP), false negative (FN), or true negative (TN). Five distinct metrics are calculated using this data: precision, recall, F1-score, Intersection over Union (IoU), and accuracy. These metrics are expressed by

$$\text{Precision} = \frac{TP}{TP+FP}, \tag{3.1}$$

$$\text{Recall} = \frac{TP}{TP+FN}, \tag{3.2}$$

$$\text{F1-Score} = \frac{2 \cdot TP}{2 \cdot TP+FP+FN}, \tag{3.3}$$

$$\text{IoU} = \frac{TP}{TP+FP+FN}, \tag{3.4}$$

e

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}. \tag{3.5}$$

3.4 Results

The results related to the occupation of hardware resources, time processing, and segmentation validation are provided in this section. Through them, the feasibility of real-time worm segmentation using the designed architecture is evaluated.

3.4.1 Hardware Area Occupation Analysis

The results related to hardware resource occupation are presented in Table 3.1. The first and second columns indicate the number of bits used to represent the image pixels (b) and the decimal part of the architecture components (d). And the third to the fifth columns indicate the number of logical cells occupied (n_{LC}), the number of multipliers implemented using DSP blocks (n_{Mult}), and the number of block memory bits (n_{BitsM}), respectively. The percentage of resource utilization available on the target FPGA is also provided.

Based on the data in Table 3.1, Figure 3.3 displays the surface that best relates the values of b, d, and n_{LC}. The equation associated with this surface, obtained by regression analysis ($R^2 = 0.9715$), can be expressed as

$$f_{n_{LC}}(b,d) = 4 \cdot 10^6 - 1.1 \cdot 10^6 \cdot b - 6.8 \cdot 10^4 \cdot d + 7.2 \cdot 10^4 \cdot b^2 + 1.1 \cdot 10^4 \cdot b \cdot d \tag{3.6}$$

Table 3.1: Hardware area occupation results for different values of b and d.

b	d	n_{LC}	n_{Mult}	n_{BitsM}
8	24	476,484 (56%)	1,518 (100%)	2,162,688 (4%)
8	19	314,846 (37%)	1,518 (100%)	1,835,008 (3%)
8	14	244194 (29%)	1,518 (100%)	1,507,328 (3%)
7	24	146,242 (17%)	1,518 (100%)	1,081,344 (2%)
7	19	131,348 (15%)	936 (62%)	917,504 (2%)
7	14	120,704 (14%)	936 (62%)	753,664 (1%)
6	24	70,432 (8%)	768 (51%)	540,672 (<1%)
6	19	63,898 (7%)	448 (30%)	458,752 (<1%)
6	14	58,662 (7%)	448 (30%)	376,832 (<1%)

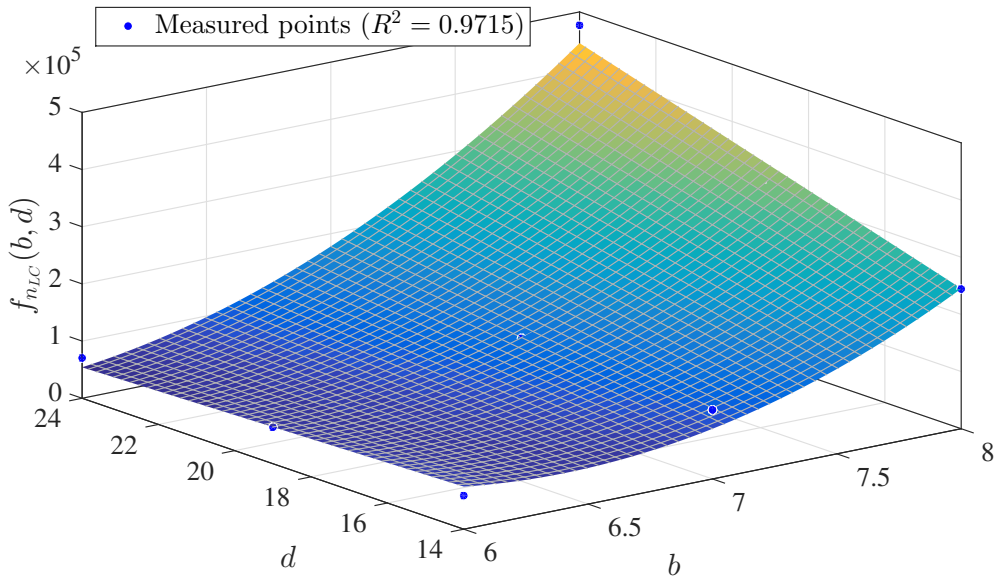

Figure 3.3: Surface, $f_{n_{LC}}(b,d)$, found to estimate the number of logical cells occupied, n_{NLC}, as a function of the number of bits b and d.

Observing the data presented in Table 3.1 and the surface shown in Figure 3.3 it is possible to conclude that the values of n_{LC} are directly proportional to b and d. The d value determines the number of bits used in the decimal part of the architecture components, so reducing this value implies using fewer hardware resources to implement the components. This variable has a linear relationship with n_{LC}. Meanwhile, the value of b impacts the number of components employed by the architecture, since the designed modules have their parallelism levels related to the number of bits used for pixel representation. Each reduced bit in the representation of the image pixels decreases the number

of components employed in the architecture by approximately half. This relationship can be better observed in the data presented for n_{BitsM}. Thus, this variable has an exponential relationship with n_{LC}.

The lower hardware resources usage allows for increased parallelism of the architecture, enabling the use of more PNH modules, or implementation of additional logic. Moreover, it is possible to implement the system on commercial FPGAs with fewer resources available.

3.4.2 Time Processing Analysis

Table 2.4 shows the results associated with the time processing. The first and second columns indicate the number of bits used to represent the image pixels (b) and the decimal part of the architecture components (d), respectively. The third gives the clock (Clk) at which the circuit operates, adopting the maximum frequency of each implementation. The fourth displays the image processing time (t_I), calculated according to Equation 2.13. The fifth presents the initial system latency (D), obtained according to Equation 2.11. And the sixth provides the architecture throughput (th) in IPS, determined through the Equation 2.12.

Table 3.2: Time processing results for the different values of b and d.

b	d	Clk (ns)	t_I (ms)	D (ms)	th (IPS)
8	24	6.04	5.56	22.24	179
8	19	4.86	4.48	17.92	223
8	14	4.63	4.27	17.08	234
7	24	5.87	5.41	21.64	184
7	19	4.68	4.31	17.24	232
7	14	4.45	4.10	16.40	243
6	24	5.56	5.12	20.48	195
6	19	4.58	4.22	16.88	236
6	14	4.44	4.09	16.36	244

Through the data in Table 2.4, the surface that best relates the throughput to the values of b and d was obtained, shown in Figure 3.4. The equation associated with this surface, determined by regression analysis ($R^2 = 0.9961$), can be expressed as

$$f_{th}(b,d) = 91.3 - 0.8 \cdot b + 22.8 \cdot d - 0.3 \cdot b \cdot d - 0.7 \cdot d^2 \tag{3.7}$$

The results presented in Table 3.2 and Figure 3.4 show that the designed architecture can achieve high throughputs in the segmentation of images with the resolution of the videos available in the public dataset (McDermott-Rouse* et al. 2021*b*). Analyzing the exposed data, it is observed that the reduction of the values of b and d provide an increase in throughput. It happens because the lower b and d, the fewer bits need to have their values computed, consequently reducing the critical time of the system.

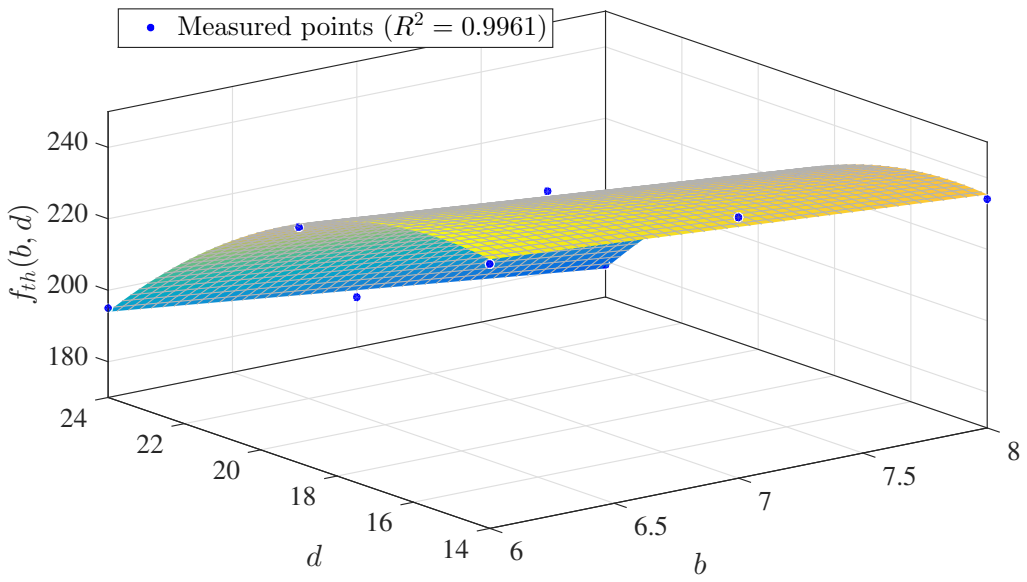

Figure 3.4: Surface, $f_{th}(b,d)$, found to estimate the throughput, th, as a function of the number of bits b and d.

The videos recorded with the structure presented in Barlow et al. (2021) are captured at a rate of 25 FPS, thus obtaining one frame every 40 ms. According to the data presented in Table 3.2, the longest image processing time, t_I, was obtained for $b = 8$ and $d = 24$, as equal to 5.56 ms. And the smallest value of t_I was obtained for $b = 6$ and $d = 14$, as equal to 4.09 ms. Considering the time between the capture of each frame equal to 40 ms, the highest and lowest value of t_I use, respectively, 13.9% and 10.2% of this time to perform the image segmentation. Therefore, the designed architecture can perform the image segmentation in real-time, even in the case of the largest t_I value. In addition, the low utilization of the available time between frame captures allows the development of more processes in real-time, such as the execution of more system algorithms or processing several videos on the same architecture. Thus, it is possible to perform real-time segmentation of the images from the five structures in the Behavioural Genomics lab using a single implementation on FPGA. It demonstrates the feasibility of developing a high throughput worm tracking system in hardware.

3.4.3 Segmentation Analysis

The validation of the image segmentation using the designed architecture was realized for different values of b and d. The images of the worms, extracted from a video available in the dataset, were segmented by hardware implementation to obtain the masks used in this evaluation. Examples of the masks obtained are shown in Figures 3.5, 3.6 and 3.7.

The 1800 images used for segmentation validation had masks generated by the designed hardware and software. The obtained masks were compared pixel by pixel to define the values of TP, FP, FN, and TN. These data are presented in the confusion matri-

(a) (b) (c)

Figure 3.5: Result of image segmentation with $b = 8$ and different values of d. (a) $d = 14$. (b) $d = 19$. (c) $d = 24$.

(a) (b) (c)

Figure 3.6: Result of image segmentation with $b = 7$ and different values of d. (a) $d = 14$. (b) $d = 19$. (c) $d = 24$.

(a) (b) (c)

Figure 3.7: Result of image segmentation with $b = 6$ and different values of d. (a) $d = 14$. (b) $d = 19$. (c) $d = 24$.

ces illustrated in Figures 3.8, 3.9, and 3.10 In these representations, the numbers 1 and 2 indicate the pixels classified as the worm region and image background, respectively.

Table 3.3 shows the results related to the validation of the image segmentation in hardware. The first and second columns indicate the number of bits used to represent the image pixels (b) and the decimal part of the architecture components (d), respectively. The third shows the precision of the obtained results, calculated according to 3.1. The fourth presents the recall values, calculated by Equation 3.2. The fifth displays the F1-Score values, obtained according to Equation 3.3. The sixth shows the IoU values, determined using Equation 3.4. And the seventh gives the accuracy values obtained according to Equation 3.5.

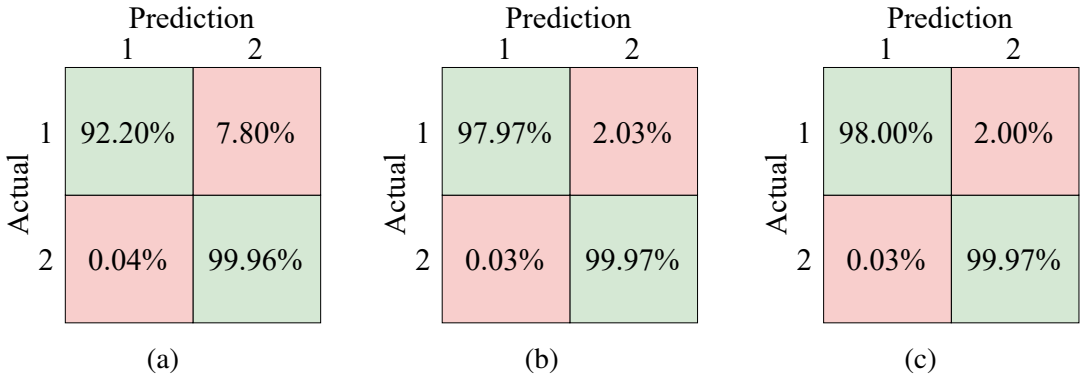

Figure 3.8: Confusion matrix for $b = 8$ and different values of d. (a) $d = 14$. (b) $d = 19$. (c) $d = 24$.

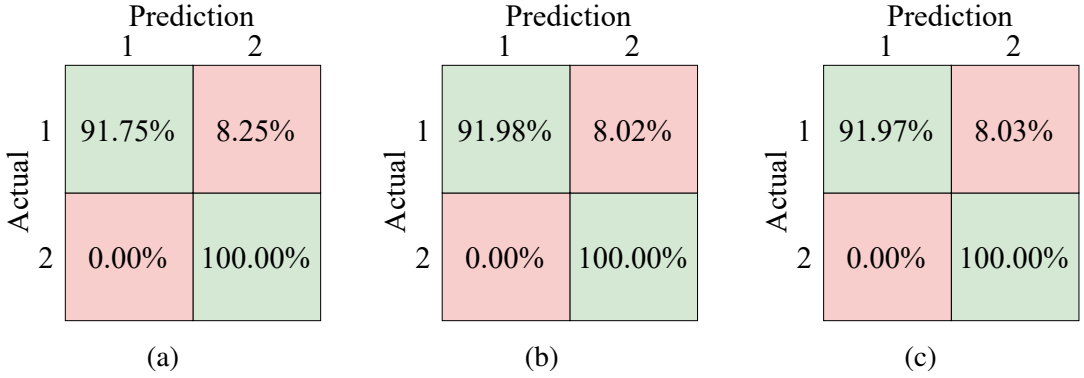

Figure 3.9: Confusion matrix for $b = 7$ and different values of d. (a) $d = 14$. (b) $d = 19$. (c) $d = 24$.

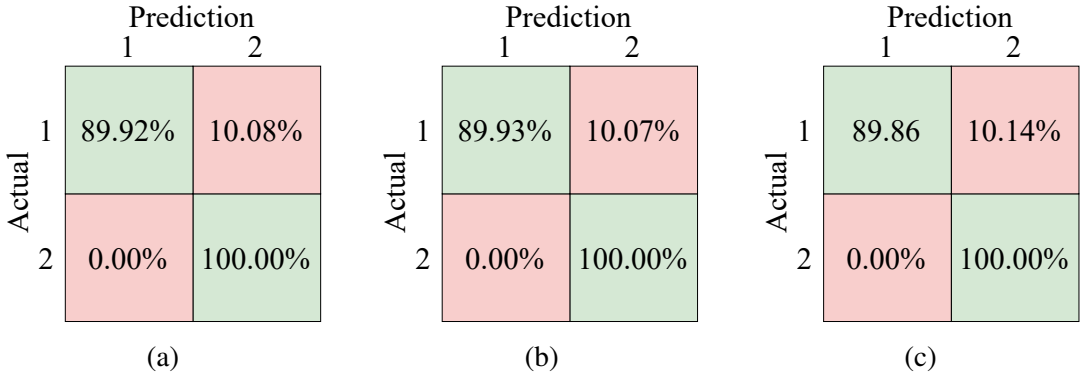

Figure 3.10: Confusion matrix for $b = 6$ and different values of d. (a) $d = 14$. (b) $d = 19$. (c) $d = 24$.

Based on the data in Table 3.3, Figure 3.11 presents the surface that relates IoU to the values of b and d. The equation associated with this surface, obtained by regression

Table 3.3: Image segmentation validation results as a function of different values of b and d.

b	d	Precision	Recall	F1-Score	IoU	Accuracy
8	24	0.9862	0.9800	0.9831	0.9667	0.9994
8	19	0.9851	0.9797	0.9824	0.9654	0.9993
8	14	0.9779	0.9220	0.9491	0.9032	0.9982
7	24	1.000	0.9197	0.9582	0.9197	0.9985
7	19	1.000	0.9198	0.9582	0.9198	0.9985
7	14	1.000	0.9175	0.9570	0.9175	0.9985
6	24	1.000	0.8986	0.9466	0.8986	0.9981
6	19	1.000	0.8993	0.9470	0.8993	0.9981
6	14	1.000	0.8992	0.9469	0.8992	0.9981

analysis ($R^2 = 0.8810$), is expressed by

$$f_{IoU}(b,d) = 0.93 + 0.02 \cdot b + 0.01 \cdot d + 0.01 \cdot b \cdot d - 0.01 \cdot d^2 \tag{3.8}$$

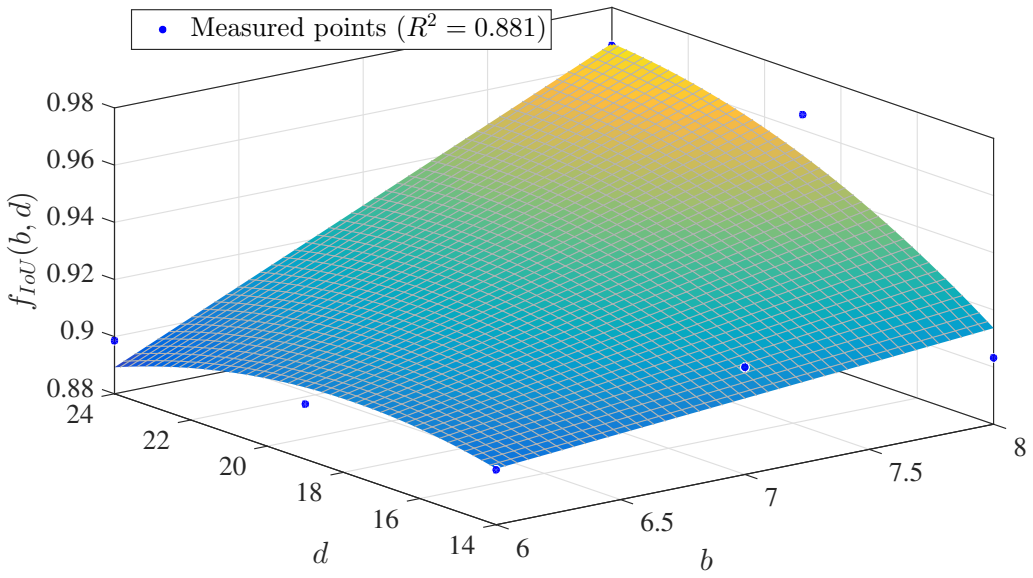

Figure 3.11: Surface, $f_{IoU}(b,d)$, found to estimate the Intersection over Union, IoU, as a function of the number of bits b and d.

Through the analysis of the results shown in Table 3.3 and Figure 3.11, it is possible to observe the reduction of Recall, F1-Score, IoU, and accuracy values along with the b and d values. It occurs because the decrease of d causes a reduction in the precision of the representation of the values, increasing the error in the results. And decreasing b reduces

the possibilities of threshold representation, causing an approximate threshold value to be obtained. However, even with the smallest values of b and d, the results obtained can still be considered satisfactory, since, as shown in Figures 3.5, 3.6 and 3.7, the worms can be identified in all the images. The presented images have some noise misclassified as worm region, but this noise is removed later by filters in the other image processing steps adopted in the worm tracking system.

3.5 Conclusions

This chapter presented an analysis of the feasibility of segmenting worm images in real-time using the Otsu method implemented in FPGA. The hardware performance was analyzed using different numbers of bits in the decimal part of the architecture components and the representation of image pixels. Results related to the occupation of hardware resources, time processing, and segmentation validation were presented. The validation was performed using high-resolution images of the worms extracted from videos available on a public dataset. The results indicated that the designed architecture performed the segmentation satisfactorily and met the system's time requirements, enabling real-time processing.

Chapter 4

Conclusion

This dissertation proposed the development of Otsu's method in FPGA for application in real-time worm image segmentation. Initially, contextualization of the theme was performed, explaining the applications of worm tracking systems and the need to achieve high processing speeds, indicating the reasons for using reconfigurable computing. Subsequently, the main contributions of this dissertation were highlighted as the development of a hardware architecture with a fully parallel scheme for Otsu's method and the demonstration of the ability to segment the worm images in real-time using this implementation.

Subsequently, the details of the architecture designed for the Otsu method were presented, with a fully parallel scheme and fixed-point representation. In the proposed implementation, the histogram computation module can have its parallelism level adjusted. Thus, synthesis results were presented for three different FPGAs, in which the histogram module was implemented with the maximum parallelism allowed by the amount of hardware resources available. All results indicated the achievement of high throughput values, demonstrating the possibility of processing high-resolution images in real-time. The throughput reached in the processing of 4K resolution images ranged between 90 and 2,057 IPS, while for 16K resolution images these values varied between 5 and 128 IPS. Compared to works in the literature, the designed architecture demonstrated that the full parallelization performed well concerning speedup and power consumption reduction. These gains were obtained by further exploiting the parallelism of the algorithm and using fixed-point representation, as well as reducing the bottleneck of the between-class variance calculation by using LUTs. The speedups achieved ranged between $20\times$ and $10^3\times$, while power consumption was reduced between $23\times$ and $10^5\times$.

Following this, the results obtained by applying the designed architecture to worm image segmentation were analyzed. This analysis was performed considering the processing of images extracted from high-resolution videos of the worms captured at a rate of 25 FPS. All results were obtained for different numbers of bits employed in the decimal part of the architecture components and the representation of the image pixels. Hardware resource occupancy and time processing data were presented with the synthesis of the architecture for the Arria 10 GX 1150 FPGA. The presented values indicated the range of throughputs between 179 and 244 IPS. With these throughputs, a maximum of 13.9% of the available time between the capture of each frame of the video is used, making it possible to perform the segmentation in real-time. The validation of the segmentation was done by comparing the results obtained in the hardware architecture to those obtained by software, present-

ing data about precision, recall, F1-Score, IoU, and accuracy. In addition, images were shown as examples of the hardware segmentation results, indicating good segmentation of the worms. The IoU values achieved ranged between 0.96 and 0.89.

Thus, it is possible to conclude that the hardware proposed in this dissertation can perform real-time segmentation of high-resolution videos from worm tracking systems. According to the data presented, the high throughput achieved allows the implementation of other processes to be executed within the real-time requirements of the system. These data indicate the feasibility of obtaining a high throughput worm tracking system through the development of dedicated hardware.

4.1 Future Works

For future work, we intend to compare the results achieved with those obtained through the Tierpsy Tracker software on GPU, seeking to understand the throughput gains provided by the hardware implementation compared to the one currently used. In addition, we should study the bit configurations that allow obtaining segmentation results with the appropriate precision for the tracking system.

The system must be implemented at the image acquisition location for real-time tests. Thus, it will be necessary to develop the communication between the FPGA and the cameras to provide the images directly to the hardware architecture.

Finally, aiming to develop a hardware framework to perform several processes and analyses in real-time, there should be efforts to implement in hardware more algorithms used in the worm tracking system. The development of fully parallel implementations that allow high throughputs must be explored.

Bibliography

Alhamwi, Ali, Bertrand Vandeportaele & Jonathan Piat (2015), Real Time Vision System for Obstacle Detection and Localization on FPGA, *em* L.Nalpantidis, V.Krüger, J.-O.Eklundh & A.Gasteratos, eds., 'Computer Vision Systems', Springer International Publishing, Cham, pp. 80–90.

Antoshechkin, Igor & Paul W. Sternberg (2007), 'The versatile worm: genetic and genomic resources for caenorhabditis elegans research', *Nature Reviews Genetics* **8**(7), 518–532.
URL: *https://doi.org/10.1038/nrg2105*

Baek, J. H., P. Cosman, Z. Feng, J. Silver & W. R. Schafer (2002), 'Using machine vision to analyze and classify Caenorhabditis elegans behavioral phenotypes quantitatively', *J Neurosci Methods* **118**(1), 9–21.

Barlow, Ida, Luigi Feriani, Eleni Minga, Adam McDermott-Rouse, Thomas O'Brien, Ziwei Liu, Maximilian Hofbauer, John R. Stowers, Erik C. Andersen, Siyu Serena Ding & André E.X. Brown (2021), 'Megapixel camera arrays for high-resolution animal tracking in multiwell plates', *bioRxiv* .
URL: *https://www.biorxiv.org/content/early/2021/07/13/2021.04.16.440222*

Bornhorst, Julia, Eike Jannik Nustede & Sebastian Fudickar (2019), 'Mass surveilance of c. elegans—smartphone-based diy microscope and machine-learning-based approach for worm detection', *Sensors* **19**(6).
URL: *https://www.mdpi.com/1424-8220/19/6/1468*

Brown, André E. X., Eviatar I. Yemini, Laura J. Grundy, Tadas Jucikas & William R. Schafer (2013), 'A dictionary of behavioral motifs reveals clusters of genes affecting caenorhabditis elegans locomotion', *Proceedings of the National Academy of Sciences* **110**(2), 791–796.
URL: *https://www.pnas.org/content/110/2/791*

Brown, André E. X. & William R. Schafer (2015), *Automated behavioural fingerprinting of Caenorhabditis elegans mutants*, Cambridge Series in Systems Genetics, Cambridge University Press, capítulo 12, p. 234–256.

Coutinho, Maria GF, Matheus F Torquato & Marcelo AC Fernandes (2019), 'Deep neural network hardware implementation based on stacked sparse autoencoder', *IEEE Access* **7**, 40674–40694.

Cronin, C. J., Z. Feng & W. R. Schafer (2006), 'Automated imaging of C. elegans behavior', *Methods Mol Biol* **351**, 241–251.

Da Costa, Alexandre LX, Caroline AD Silva, Matheus F Torquato & Marcelo AC Fernandes (2019), 'Parallel implementation of particle swarm optimization on fpga', *IEEE Transactions on Circuits and Systems II: Express Briefs* **66**(11), 1875–1879.

Das, R. K., A. De, C. Pal & A. Chakrabarti (2014), DSP hardware design for fingerprint binarization and thinning on FPGA, *em* 'Proceedings of The 2014 International Conference on Control, Instrumentation, Energy and Communication (CIEC)', pp. 544–549.

Dias, Leonardo A, Augusto MP Damasceno, Elena Gaura & Marcelo AC Fernandes (2021), 'A full-parallel implementation of self-organizing maps on hardware', *Neural Networks* .

Dusenbery, David B. (1985), 'Using a microcomputer and video camera to simultaneously track 25 animals', *Computers in Biology and Medicine* **15**(4), 169–175.
URL: *https://www.sciencedirect.com/science/article/pii/0010482585900587*

Evans, Kathryn S., Marijke H. van Wijk, Patrick T. McGrath, Erik C. Andersen & Mark G. Sterken (2021), 'From qtl to gene: C. elegans facilitates discoveries of the genetic mechanisms underlying natural variation', *Trends in Genetics* **37**(10), 933–947.
URL: *https://www.sciencedirect.com/science/article/pii/S0168952521001463*

Faumont, Serge, Gary Rondeau, Tod R. Thiele, Kristy J. Lawton, Kathryn E. McCormick, Matthew Sottile, Oliver Griesbeck, Ellie S. Heckscher, William M. Roberts, Chris Q. Doe & Shawn R. Lockery (2011), 'An image-free opto-mechanical system for creating virtual environments and imaging neuronal activity in freely moving caenorhabditis elegans', *PLOS ONE* **6**(9), 1–12.
URL: *https://doi.org/10.1371/journal.pone.0024666*

Feng, Zhaoyang, Christopher J. Cronin, John H. Wittig, Paul W. Sternberg & William R. Schafer (2004), 'An imaging system for standardized quantitative analysis of c. elegans behavior', *BMC Bioinformatics* **5**(1), 115.
URL: *https://doi.org/10.1186/1471-2105-5-115*

Giunti, Sebastián, Natalia Andersen, Diego Rayes & María José De Rosa (2021), 'Drug discovery: Insights from the invertebrate caenorhabditis elegans', *Pharmacology Research & Perspectives* **9**(2), e00721.
URL: *https://bpspubs.onlinelibrary.wiley.com/doi/abs/10.1002/prp2.721*

Gokhale, M.B. & P.S. Graham (2005), *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*, Springer, Dordrecht.

Gonzalez, R. C. & R. E. Woods (2018), *Digital Image Processing*, 4ª edição, Pearson, New York.

HajiRassouliha, Amir, Andrew J. Taberner, Martyn P. Nash & Poul M.F. Nielsen (2018), 'Suitability of recent hardware accelerators (dsps, fpgas, and gpus) for computer vision and image processing algorithms', *Signal Processing: Image Communication* **68**, 101 – 119.
URL: *http://www.sciencedirect.com/science/article/pii/S0923596518303606*

Hebert, Laetitia, Tosif Ahamed, Antonio C. Costa, Liam O'Shaughnessy & Greg J. Stephens (2021), 'Wormpose: Image synthesis and convolutional networks for pose estimation in c. elegans', *PLOS Computational Biology* **17**(4), 1–20.
URL: *https://doi.org/10.1371/journal.pcbi.1008914*

Husson, S. J., W. S. Costa, C. Schmitt & A. Gottschalk (2013), 'Keeping track of worm trackers', *WormBook* pp. 1–17.

Intel (2020*a*), 'Intel® Agilex™ FPGAs and SoCs device overview', Available: `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/agilex/ag-overview.pdf`. Accessed: 22 jan. 2021.

Intel (2020*b*), 'Intel® Stratix® 10 GX/SX device overview', Available: `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10-overview.pdf`. Accessed: 22 jan. 2021.

Javer, Avelino, Michael Currie, Chee Wai Lee, Jim Hokanson, Kezhi Li, Céline N. Martineau, Eviatar Yemini, Laura J. Grundy, Chris Li, QueeLim Ch'ng, William R. Schafer, Ellen A. A. Nollen, Rex Kerr & André E. X. Brown (2018), 'An open-source platform for analyzing and sharing worm-behavior data', *Nature Methods* **15**(9), 645–646.
URL: *https://doi.org/10.1038/s41592-018-0112-1*

Jianlai, Wang, Yang Chunling, Zhu Min & Wang Changhui (2009), Implementation of Otsu's thresholding process based on FPGA, *em* '2009 4th IEEE Conference on Industrial Electronics and Applications', pp. 479–483.

Kim, H. S. (2018), 'FPGA-based of thermogram enhancement algorithm for non-destructive thermal characterization', *International Journal of Engineering* **31**(10), 1675–1681.

Kim, Wooseong, Gabriel Lambert Hendricks, Kiho Lee & Eleftherios Mylonakis (2017), 'An update on the use of c. elegans for preclinical drug discovery: screening and identifying anti-infective drugs', *Expert Opinion on Drug Discovery* **12**(6), 625–633. PMID: 28402221.
URL: *https://doi.org/10.1080/17460441.2017.1319358*

Koopman, Mandy, Quentin Peter, Renée I. Seinstra, Michele Perni, Michele Vendruscolo, Christopher M. Dobson, Tuomas P. J. Knowles & Ellen A. A. Nollen (2020), 'Assessing motor-related phenotypes of caenorhabditis elegans with the wide field-of-view nematode tracking platform', *Nature Protocols* **15**(6), 2071–2106.
URL: *https://doi.org/10.1038/s41596-020-0321-9*

Kwon, Namseop, Jaeyeon Pyo, Seung-Jae Lee & Jung Ho Je (2013), '3-d worm tracker for freely moving c. elegans', *PLOS ONE* **8**(2), 1–6.
URL: *https://doi.org/10.1371/journal.pone.0057484*

Ladner, Richard E. & Michael J. Fischer (1980), 'Parallel prefix computation', *J. ACM* **27**(4), 831–838.
URL: *https://doi.org/10.1145/322217.322232*

Layana Castro, Pablo E., Joan Carles Puchalt, Antonio García Garví & Antonio-José Sánchez-Salmerón (2021), 'Caenorhabditis elegans multi-tracker based on a modified skeleton algorithm', *Sensors* **21**(16).
URL: *https://www.mdpi.com/1424-8220/21/16/5622*

Leifer, Andrew M., Christopher Fang-Yen, Marc Gershow, Mark J. Alkema & Aravinthan D. T. Samuel (2011), 'Optogenetic manipulation of neural activity in freely moving caenorhabditis elegans', *Nature Methods* **8**(2), 147–152.
URL: *https://doi.org/10.1038/nmeth.1554*

Mathew, Mark D., Neal D. Mathew & Paul R. Ebert (2012), 'Wormscan: A technique for high-throughput phenotypic analysis of caenorhabditis elegans', *PLOS ONE* **7**(3), 1–6.
URL: *https://doi.org/10.1371/journal.pone.0033483*

Maulik, Malabika, Swarup Mitra, Abel Bult-Ito, Barbara E. Taylor & Elena M. Vayndorf (2017), 'Behavioral phenotyping and pathological indicators of parkinson's disease in c. elegans models', *Frontiers in Genetics* **8**, 77.
URL: *https://www.frontiersin.org/article/10.3389/fgene.2017.00077*

McCool, Michael, Arch D. Robison & James Reinders (2012), Chapter 2 - background, *em* 'Structured Parallel Programming', Morgan Kaufmann, Boston, pp. 39–75.

McDermott-Rouse, Adam, Eleni Minga, Ida Barlow, Luigi Feriani, Philippa H Harlow, Anthony J Flemming & André E X Brown (2021*a*), 'Behavioral fingerprints predict insecticide and anthelmintic mode of action', *Molecular Systems Biology* **17**(5), e10267.
URL: *https://www.embopress.org/doi/abs/10.15252/msb.202110267*

McDermott-Rouse*, Adam, Eleni Minga*, Ida Barlow, Luigi Feriani, Philippa H Harlow, Anthony J Flemming & André EX Brown (2021*b*), '20191205_rr01_sp01_ds02 [data set]', Zenodo. `https://doi.org/10.5281/zenodo.4671872`.

McDiarmid, T. A., A. J. Yu & C. H. Rankin (2018), 'Beyond the response—high throughput behavioral analyses to link genome to phenome in caenorhabditis elegans', *Genes, Brain and Behavior* **17**(3), e12437. e12437 G2B-00116-2017.R1.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1111/gbb.12437*

Moore, Brad T., James M. Jordan & L. Ryan Baugh (2013), 'Wormsizer: High-throughput analysis of nematode size and shape', *PLOS ONE* **8**(2), 1–13.
URL: *https://doi.org/10.1371/journal.pone.0057142*

Moy, Kyle, Weiyu Li, Huu Phuoc Tran, Valerie Simonis, Evan Story, Christopher Brandon, Jacob Furst, Daniela Raicu & Hongkyun Kim (2016), 'Computational methods for tracking, quantitative assessment, and visualization of c. elegans locomotory behavior', *PLOS ONE* **10**(12), 1–22.
URL: *https://doi.org/10.1371/journal.pone.0145870*

Nigon, Victor M. & Marie-Anne Félix (2017), 'History of research on c. elegans and other free-living nematodes as model organisms', *WormBook* pp. 1–84.
URL: *https://doi.org/10.1895/wormbook.1.181.1*

O'Reilly, Linda P., Cliff J. Luke, David H. Perlmutter, Gary A. Silverman & Stephen C. Pak (2014), 'C. elegans in high-throughput drug discovery', *Advanced Drug Delivery Reviews* **69-70**, 247–253. Innovative tissue models for drug discovery and development.
URL: *https://www.sciencedirect.com/science/article/pii/S0169409X13002871*

Otsu, N. (1979), 'A threshold selection method from gray-level histograms', *IEEE Transactions on Systems, Man, and Cybernetics* **9**(1), 62–66.

Pandey, J. G., A. Karmakar, C. Shekhar & S. Gurunarayanan (2014), A Novel Architecture for FPGA Implementation of Otsu's Global Automatic Image Thresholding Algorithm, *em* '2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems', pp. 300–305.

Pandey, Jai Gopal & Abhijit Karmakar (2019), 'Unsupervised image thresholding: hardware architecture and its usage for FPGA-SoC platform', *International Journal of Electronics* **106**(3), 455–476.
URL: *https://doi.org/10.1080/00207217.2018.1540065*

Perni, Michele, Pavan K. Challa, Julius B. Kirkegaard, Ryan Limbocker, Mandy Koopman, Maarten C. Hardenberg, Pietro Sormanni, Thomas Müller, Kadi L. Saar, Lianne W.Y. Roode, Johnny Habchi, Giulia Vecchi, Nilumi Fernando, Samuel Casford, Ellen A.A. Nollen, Michele Vendruscolo, Christopher M. Dobson & Tuomas P.J. Knowles (2018), 'Massively parallel c. elegans tracking provides multi-dimensional fingerprints for phenotypic discovery', *Journal of Neuroscience Methods* **306**, 57–67.
URL: *https://www.sciencedirect.com/science/article/pii/S016502701830027X*

Ramot, Daniel, Brandon E. Johnson, Tommie L. Berry, Jr, Lucinda Carnell & Miriam B. Goodman (2008), 'The parallel worm tracker: A platform for measuring average speed and drug-induced paralysis in nematodes', *PLOS ONE* **3**(5), 1–7.
URL: *https://doi.org/10.1371/journal.pone.0002208*

Ren, X. & Y. Wang (2016), Design of a FPGA hardware architecture to detect real-time moving objects using the background subtraction algorithm, *em* '2016 5th International Conference on Computer Science and Network Technology (ICCSNT)', pp. 428–433.

Risse, Benjamin, Dimitri Berh, Nils Otto, Christian Klämbt & Xiaoyi Jiang (2017), 'Fimtrack: An open source tracking and locomotion analysis software for small animals', *PLOS Computational Biology* **13**(5), 1–15.
URL: *https://doi.org/10.1371/journal.pcbi.1005530*

Roussel, Nicolas, Jeff Sprenger, Susan J. Tappan & Jack R. Glaser (2015), 'Robust tracking and quantification of c. elegans body shape and locomotion through coiling, entanglement, and omega bends', *Worm* **3**(4), e982437–e982437.
URL: *https://pubmed.ncbi.nlm.nih.gov/26435884*

Silva, Sérgio N, Felipe F Lopes, Carlos Valderrama & Marcelo AC Fernandes (2020), 'Proposal of takagi–sugeno fuzzy-pi controller hardware', *Sensors* **20**(7), 1996.

Swierczek, Nicholas A., Andrew C. Giles, Catharine H. Rankin & Rex A. Kerr (2011), 'High-throughput behavioral analysis in c. elegans', *Nature Methods* **8**(7), 592–598.
URL: *https://doi.org/10.1038/nmeth.1625*

Tian, H., S. K. Lam & T. Srikanthan (2003), Implementing Otsu's thresholding process using area-time efficient logarithmic approximation unit, *em* 'Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.', Vol. 4, pp. IV–IV.

Torquato, Matheus F & Marcelo AC Fernandes (2019), 'High-performance parallel implementation of genetic algorithm on fpga', *Circuits, Systems, and Signal Processing* **38**(9), 4014–4039.

Torres-Monsalve, A. F. & J. Velasco-Medina (2016), Hardware implementation of ISODATA and Otsu thresholding algorithms, *em* '2016 XXI Symposium on Signal Processing, Images and Artificial Vision (STSIVA)', pp. 1–5.

Tsechpenakis, Gabriel, Laura Bianchi, Dimitris N. Metaxas & Monica Driscoll (2008), 'A novel computational approach for simultaneous tracking and feature extraction of c. elegans populations in fluid environments', *IEEE Transactions on Biomedical Engineering* **55**(5), 1539–1549.

Tsibidis, G. D. & N. Tavernarakis (2007), 'Nemo: a computational tool for analyzing nematode locomotion', *BMC Neurosci* **8**, 86.

Tulasigeri, C. & M. Irulappan (2016), An advanced thresholding algorithm for diagnosis of glaucoma in fundus images, *em* '2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)', pp. 1676–1680.

Vahid, F. (2010), *Digital Design with RTL Design, Verilog and VHDL*, 2ª edição, John Wiley & Sons, Massachusetts.

Wang, W. & X. Huang (2013), An FPGA co-processor for adaptive lane departure warning system, *em* '2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)', pp. 1380–1383.

Wang, Wenhui, Yu Sun, Scott J. Dixon, Mariam Alexander & Peter J. Roy (2009), 'An automated micropositioning system for investigating c. elegans locomotive behavior', *JALA: Journal of the Association for Laboratory Automation* **14**(5), 269–276. **URL:** *https://doi.org/10.1016/j.jala.2008.12.006*

White, Amelia G., Brandon Lees, Huey-Ling Kao, P. Giselle Cipriani, Eliana Munarriz, Annalise B. Paaby, Katherine Erickson, Sherly Guzman, Kirk Rattanakorn, Eduardo Sontag, Davi Geiger, Kristin C. Gunsalus & Fabio Piano (2013), 'Devstar: High-throughput quantification of c. elegans developmental stages', *IEEE Transactions on Medical Imaging* **32**(10), 1791–1803.

Xilinx (2006), 'Xcell journal: Virtex-5 special edition', Available: `https://www.xilinx.com/publications/archives/xcell/Xcell59.pdf`. Accessed: 24 feb. 2021.

Xilinx (2010), 'Spartan-6 FPGA configurable logic block: User guide', Available: `https://www.xilinx.com/support/documentation/user_guides/ug384.pdf`. Accessed: 24 feb. 2021.

Yemini, E. (2013), High-throughput, single-worm tracking and analysis in Caenorhabditis elegans, Ph.d. thesis, University of Cambridge.

Zhao, J., Bingqian Xie & X. Huang (2014), Real-time lane departure and front collision warning system on an FPGA, *em* '2014 IEEE High Performance Extreme Computing Conference (HPEC)', pp. 1–5.