Proposta de Implementação em Hardware do Algoritmo de Otsu Aplicado ao Rastreamento em Tempo Real de Vermes

Wysterlânya Kyury Pereira Barros
Universidade Federal do Rio Grande do Norte – UFRN
kyurybarros@gmail.com

Marcelo A. C. Fernandes
Universidade Federal do Rio Grande do Norte – UFRN
mfernandes@dca.ufrn.br

ABSTRACT

This work proposes an implementation in Field Programmable Gate Array (FPGA) of the Otsu's method applied to real-time tracking of worms called *Caenorhabditis elegans*. Real-time tracking is necessary to measure changes in the worm's behavior in response to treatment with Ribonucleic Acid (RNA) interference. Otsu's method is a global thresholding algorithm used to define an optimal threshold between two classes. However, this technique in real-time applications associated with the processing of high-resolution videos has a high computational cost because of the massive amount of data generated. Otsu's algorithm needs to identify the worms in each frame captured by a high-resolution camera in a real-time analysis of the worm's behavior. Thus, this work proposes a high-performance implementation of Otsu's algorithm in FPGA. The results show it was possible to achieve a speedup up to 5 times higher than similar works in the literature.

KEYWORDS

Método de Otsu, Processamento de Imagem, FPGA, Implementação Paralela, Caenorhabditis Elegans

1 INTRODUÇÃO

Nos últimos anos, houve um aumento na quantidade de soluções computacionais empregando técnicas de Processamento Digital de Imagens (PDI). Dentre essas, algumas necessitam de processamento em tempo real, tais como reconhecimento facial, aprimoramento de imagens médicas, autenticação de assinaturas, controle de tráfego, carros autônomos, análise da qualidade de produtos, entre outras. Atender ao requisito de tempo dessas aplicações pode ser algo complexo. Dependendo da resolução da imagem, profundidade de cor e, no caso de vídeos, quantidade de quadros por segundo (*Frames Per Second* - FPS), é possível que o problema a ser resolvido possua um grande volume de dados, dificultando, assim, a obtenção dos resultados no período desejado.

Uma área com número crescente de pesquisas que necessita do processamento em tempo real de vídeos é a análise de sistemas biológicos. Através de imagens microscópicas vários fenômenos biológicos são observados, sendo que realizar essa observação manualmente exige muito esforço. Devido a isso, vários trabalhos na literatura utilizam métodos de PDI para automatizar esse processo, alguns exemplos de aplicações são a análise morfológica de células [1], classificação de células [2], análise do ciclo celular [3], rastreamento e classificação de micropartículas [4], detecção de células cancerígenas [5], entre outras.

Esses vídeos geralmente apresentam um volume de dados massivo a ser processado num curto intervalo de tempo, sendo a capacidade de processamento com alta velocidade um problema crítico para esses sistemas. Devido a isso, surgiu como alternativa para aceleração dos algoritmos empregados o uso de processadores gráficos (*Graphics Processing Units - GPUs*) e computação reconfigurável (*Reconfigurable Computing - RC*). A computação em nuvem, que pode oferecer recursos de GPU e RC, não consegue atender aos requisitos de tempo real desse tipo de aplicação, sendo necessário a implementação de soluções no local de aquisição dos dados.

Vários trabalhos na literatura apresentam comparações entre GPU e RC para a aceleração de algoritmos de visão computacional e processamento digital de imagem. Em [6] é apresentada uma revisão de vários artigos que fazem essa análise, concluindo-se que para aplicações que necessitam de alta velocidade de processamento é mais indicado o emprego dos RCs. A computação reconfigurável, com os dispositivos *Field-Programmable Gate Arrays* (FPGAs), permite que os algoritmos sejam paralelizados e otimizados em nível de portas lógicas para acelerar as operações, podendo assim desenvolver um hardware dedicado capaz de explorar o paralelismo inerente às imagens [7].

Diante do exposto, o presente trabalho apresenta uma proposta de implementação em FPGA do método de Otsu para a segmentação de vermes em tempo real. Este trabalho encontra-se dentro de um contexto maior, no qual está vinculado ao projeto *Quantifying behavioural phenotype space: chemistry-to-gene screens and combination therapies* (PHENOSPACE) do *Behavioral Genomics Lab*, que trabalha com a análise do comportamento do nematoide *C. elegans*. Através da observação desse verme é possível mensurar mudanças no seu comportamento em resposta ao tratamento com moléculas pequenas e interferência de ácido ribonucleico (*Ribonucleic Acid* - RNA) em todo o genoma, permitindo prever mecanismos de ação, descobrir novas associações genecomportamento e triagem de drogas [8]. Um dos objetivos desse projeto é desenvolver uma plataforma de processamento de imagem de alto desempenho para capturar sequências comportamentais complexas e interpretá-las automaticamente em tempo real. Dessa forma, este trabalho contribui para o desenvolvimento de um *framework* em hardware dedicado de alta velocidade de processamento (*high-throughput*) para rastreamento e extração de características de vermes.

2 ESTADO DA ARTE

Dentro do contexto de segmentação de imagem, muitos trabalhos na literatura apresentam aplicações de tempo real que utilizam o Método de Otsu implementado em FPGA. Em [9] é apresentado um sistema de detecção e alerta de saída de faixa adaptável; em [10] um sistema de aviso de saída de pista e colisão frontal; em [11] um sistema de visão para detecção de obstáculos e localização de um robô que navega em ambiente interno; em [12] um sistema para detecção de objetos em movimento; em [13] um sistema auxiliar no diagnóstico de glaucoma; e em [14] um sistema para aprimoramento de termogramas. Dados de ocupação dos recursos do dispositivo

FPGA e tempo de processamento relativos à implementação do método de Otsu não foram levantados nesses artigos.

O trabalho descrito em [15] foi uma das primeiras implementações do método de Otsu em FPGA. Nesse trabalho foi explorado o ganho no desempenho do algoritmo através da utilização dos recursos de computação paralela em FPGA e o emprego de MegaCores da Altera, usados para eliminar divisões e multiplicações do método de Otsu. A arquitetura proposta foi sintetizada para um dispositivo FPGA Altera Cyclone II. Os resultados apresentados expõem imagens obtidas através da segmentação em FPGA, indicando através disso o desempenho satisfatório da implementação.

Visando também eliminar divisões e multiplicações do método de Otsu, os trabalhos apresentados em [16], [17], [18] e [19] propõem a implementação da técnica com uso de funções logarítmicas. Em [16] e [18] são apresentadas duas implementações, uma da técnica direta e outra utilizando função logarítmica, sendo analisado em ambos a utilização de recursos do hardware para cada implementação. O trabalho proposto em [16] sintetizou a arquitetura desenvolvida para um dispositivo FPGA Xilinx Virtex XCV800 HQ240-4, os recursos utilizados pela implementação da forma direta foram 622 *slices* e 103 blocos de entrada/saída (input/output blocks - IOBs), enquanto a implementação com a função logarítmica utilizou 109 *slices* e 49 IOBs. Já no trabalho de [18] a arquitetura proposta foi sintetizada para um dispositivo FPGA Altera Cyclone IV EP4CE115F29C6N, em que os resultados para a implementação da técnica direta apontaram a utilização de 6525 elementos lógicos, 4920 registradores, 18266 blocos de memória e 79 multiplicadores de 9 bits, enquanto com o uso da função logarítmica foram utilizados 2440 elementos lógicos, 1026 registradores, 10943 bits de memória e 46 multiplicadores de 9 bits.

Em [17] e [19] as arquiteturas desenvolvidas são bastante semelhantes, sendo ambas implementadas em VHDL utilizando representação em ponto-fixo. Os dois projetos utilizam FPGA Xilinx Virtex-5 xc5vfx70tffg1-136-1 disponível na plataforma de desenvolvimento ML-507 da Xilinx, operando com um clock de 25,175 MHz. O artigo descrito em [17] apresentou nos resultados os recursos de hardware utilizados, apontando o uso de 168 *slices*, 33 IOBs, 4 blocos de RAM (BRAMs) e 5 multiplicadores. Também foi apresentado em [19] os recursos utilizados pela arquitetura proposta, apresentando mais dados referentes à ocupação do hardware. Além de resultados relacionados ao consumo energético e latência. Nesse foi indicado o uso de 161 *slices*, 21 IOBs, 72 LUTs, 591 registradores, 4 BRAMs e 5 multiplicadores. A potência total consumida foi de 1440,59 mW, sendo 6,48 mW de potência dinâmica e 1434,11 mW de potência estática. A latência da arquitetura foram 5 clocks, sendo aproximadamente 198,6 ns. O *throughput* alcançado foi apresentado em relação à quantidade de mega bits processados por segundo (Mbps), sendo igual a 40, 28 Mbps.

Um projeto de hardware para binarização e afinamento de imagens de impressões digitais, utilizando o método de Otsu para a binarização, é proposto em [20]. Em que a arquitetura desenvolvida foi sintetizada para um dispositivo FPGA Xilinx Spartan 6 LX45. Os recursos utilizados foram 1.898 *slice* de registradores, 1.859 *slice* de LUTs, 735 *slices*, 10 IOBs e 44 BRAMs. Essa arquitetura utilizou um clock de 100MHz, havendo uma latência de 531 clocks ou 5310 ns. O tempo de execução para o processamento de uma imagem com dimensão 280×265 e um clock de 100MHz foi igual a 1,489 ms. Nesse trabalho foi também apresentando o tempo de execução da técnica implementada em Matlab, sendo indicado que a implementação em FPGA foi aproximadamente 10× mais rápido que a versão em Matlab.

Assim, semelhante aos trabalhos apresentados na literatura, este trabalho propõe a implementação em FPGA do método de Otsu visando obter ganho de desempenho. A arquitetura desenvolvida busca acelerar a execução do algoritmo através de uma implementação totalmente paralela. A implementação do hardware é realizada utilizando design Register-Transfer Level (RTL) e representação em ponto-fixo. O dispositivo FPGA alvo é a Stratix V 5SGXMBBR3H43C3, sendo apresentado resultados de ocupação de hardware e tempo de execução para a arquitetura de hardware desenvolvida.

3 MÉTODO DE OTSU

O método de Otsu é um dos mais populares algoritmos de limiarização, utilizado para encontrar um limiar ótimo que separa duas classes, fundo e objeto de uma imagem, representadas por C_0 e C_1. Esse método apresenta como vantagem a realização de todos os seus cálculos baseado apenas no histograma da imagem [21, 22].

Inicialmente é calculado o histograma normalizado de uma imagem, **A**, em escala de cinza expressa como

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

onde $M \times N$ é a dimensão dessa imagem e $a_{i,j}$ é um pixel de b bits. O valor de cada pixel pode ser caracterizado como um nível de intensidade de cinza em uma faixa de 0 a $L-1$, onde $L = 2^b$. Cada n-ésimo pixel é processado em um instante, t_s, que é caracterizado aqui como tempo de amostragem. Assim, pode-se dizer que uma imagem completa é processada a cada m-ésimo instante, onde

$$m = M \times N \times t_s. \quad (2)$$

O histograma de cada m-ésima imagem $\mathbf{A}(m)$ é calculado e armazenado no vetor

$$\mathbf{p}(m) = [p_0(m), \ldots, p_k(m), \ldots, p_{L-1}(m)] \quad (3)$$

onde cada k-ésimo elemento $p_k(m)$ é expresso como

$$p_k(m) = \frac{n_k}{M \times N}. \quad (4)$$

onde n_k pode ser caracterizado como

$$n_k = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} c_{i,j} \quad (5)$$

e

$$c_{i,j} = \begin{cases} 1 & \text{se } a_{i,j} = k \\ 0 & \text{se } a_{i,j} \neq k \end{cases} \quad (6)$$

Após o cálculo do histograma normalizado, armazenado no vetor **p**, o método de Otsu irá calcular um limiar ótimo entre as duas

classes, ou seja, C_0 e C_1. O limiar ótimo caracterizado aqui como, $k^*(m)$, pode ser expresso como

$$k^*(m) = arg \max_{0 \leq k \leq L-1} \sigma_k^2(m). \quad (7)$$

onde $\sigma_k^2(m)$ é a k-ésima variância entre classes da m-ésima imagem, expressa como

$$\sigma_k^2(m) = \frac{(\mu_{255}(m) \times \omega_k(m) - \mu_k(m))^2}{\omega_k(m)\,(1 - \omega_k(m))}, \quad (8)$$

onde $\omega_k(m)$ e $\mu_k(m)$ são, respectivamente, a probabilidade de ocorrência de classes dado um limiar k e a intensidade média dos pixels até o nível k da m-ésima imagem. E $\mu_{255}(m)$ é a intensidade média de todos os pixels da m-ésima imagem, denominada média global, com valor igual a $\mu_k(m)$ quando $k = 255$.

As variáveis $\omega_k(m)$ e $\mu_k(m)$ podem ser expressas como

$$\omega_k(m) = \sum_{i=0}^{k} p_i(m) \quad (9)$$

e

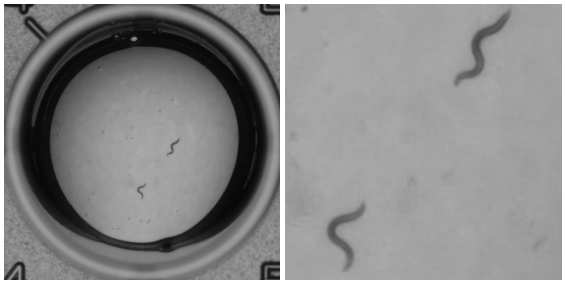
$$\mu_k(m) = \sum_{i=0}^{k} i \cdot p_i(m). \quad (10)$$

Após encontrar o valor de limiar ótimo, $k^*(m)$, no m-ésimo instante, a imagem de entrada, $\mathbf{A}(m)$, pode ser segmentada tendo seus pixels classificados como objeto ou fundo, C_0 e C_1, gerando uma máscara da imagem de entrada.

4 DESCRIÇÃO DO PROJETO

4.1 Estrutura de Análise

No *Behavioral Genomics Lab* foi desenvolvido um sistema para capturar simultaneamente imagens de diferentes vermes em diversas lâminas, permitindo a análise da mudança comportamental de cada nematoide de forma paralela. A obtenção de uma resolução satisfatória para extrair as informações comportamentais do verme foi alcançada com o emprego de uma matriz de seis câmeras capazes de gerar um vídeo de alta qualidade com 12 megapixels e taxa de 25 quadros por segundo, que captura a imagem de até 48 lâminas e centenas de vermes ao mesmo tempo [8]. A visualização aproximada de uma única lâmina é apresentada na Fig. 1a e o enquadramento apenas dos vermes presente nela na Fig. 1b.



(a) (b)

Figure 1: (a) Visualização aproximada de uma única lâmina. (b) Vermes presentes nessa lâmina.

O rastreamento e a extração de *features* dos vermes são realizados automaticamente por meio de soluções computacionais. Atualmente esses processos são efetuados *offline* por software, porém, deseja-se desenvolver uma implementação de *high-throughput* capaz de operar em tempo real. Uma maneira de alcançar essa performance é a implementação de um hardware dedicado para os algoritmos de *tracking*.

Na Fig. 2 é ilustrada uma estrutura de análise com os algoritmos de rastreamento e extração de *features* implementados em FPGA. Inicialmente, é realizada a captura das imagens microscópicas com a matriz de câmeras. As imagens são capturadas com a resolução de 4024×3026 a uma taxa de 25 *frames/s*, sendo diretamente fornecidas como entrada ao dispositivo FPGA. Ao entrar no FPGA as imagens são pré-processadas e fragmentadas em resoluções menores para a segmentação. A segmentação é realizada através do método de Otsu, podendo ser realizado o processamento paralelo das imagens fornecidas pela etapa anterior através de processos paralelos $(P_1, P_2, P_3 ... P_n)$ com a mesma arquitetura de hardware proposta nesse trabalho. A saída dessa etapa são as imagens segmentadas, com o reconhecimento da região do verme e o fundo preto. Através dessas imagens realiza-se a extração das *features* desejadas. Por fim, todas as *features* extraídas são armazenadas em disco para o acesso posterior por um computador. Nesse trabalho será apresentado o hardware dedicado desenvolvido para a etapa de segmentação dos vermes.

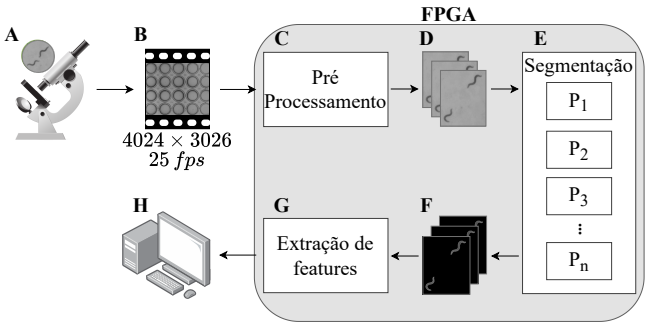


Figure 2: Estrutura para rastreamento e extração de *features* dos vermes em FPGA.

4.2 Arquitetura Proposta

4.2.1 Arquitetura Geral. A arquitetura geral da implementação em hardware do método de Otsu é apresentada através de diagrama de blocos na Fig. 3. Essa arquitetura foi desenvolvida baseada na descrição do método apresentado na Seção 3 e possui cinco módulos principais denominados de *Normalized Histogram Module* (NHM), *Probability of Class Occurrence Module* (PCOM), *Mean Intensity Module* (MIM), *Between-Class Variance Module* (BCVM) e *Optimal Threshold Module* (OTM).

Inicialmente, os pixels são fornecidos sequencialmente como entrada ao NHM, no qual é calculado o histograma normalizado da imagem de entrada, conforme as Equações 3 e 4. Posteriormente, os componentes do histograma são utilizados no PCOM, para cálculo da probabilidade de ocorrência de classes, conforme a Equação 9, e no MIM, para cálculo das intensidades médias da imagem até o valor k, conforme a Equação 10. As saídas desses dois módulos são

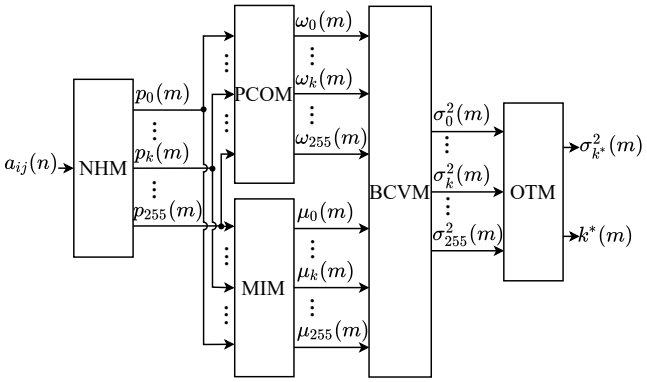


Figure 3: Arquitetura geral do projeto.

fornecidos ao BCVM, no qual é calculado o valor das variâncias entre classes, de acordo com a Equação 8. As variâncias calculadas são comparadas no OTM para selecionar o valor de limiar ótimo, conforme a Equação 7.

Todas as variáveis e constantes do projeto estão em ponto fixo. Cada n-ésimo pixel da imagem de entrada, $a_{ij}(n)$, é representado como um inteiro sem sinal de 8 bits. Já os componentes do histograma, $p_k(m)$, e a k-ésima probabilidade de ocorrência de classe, $\omega_k(m)$, são definidos como números fracionários sem sinal com apenas 24 bits na parte fracionária. A intensidade média até o k-ésimo nível, $\mu_k(m)$, utiliza 8 bits na parte inteira e 24 bits na fracionária sem bit de sinal. A k-ésima variância entre classes, $\sigma_k^2(m)$, utiliza 27 bits na parte inteira (utilizando um para sinal) e 24 bits na fracionária. Por fim, o limiar ótimo, $k^*(m)$, é representado como um inteiro sem sinal de 8 bits.

Os módulos são executados em pipeline. Todo o sistema trabalha no mesmo *sample time*, t_s, porém cada módulo possui um tempo de execução distinto, caracterizados aqui como t_1, t_2, t_3 e t_4. Para minimizar o controle, devido a falta de sincronismo entre os módulos, o hardware aqui proposto define que $t_1 = t_2 = t_3 = t_4 = t_I$, onde t_I é o tempo de processar uma imagem, sendo seu valor obtido conforme a Equação 2. Dessa forma, o sistema possui uma latência inicial expressa como $D = 3 \times t_I$ e um *throughput* caracterizado como $th = 1/t_I$ imagens por segundo (IPS).

4.2.2 Normalized Histogram Module (NHM). O NHM é responsável pela geração do histograma normalizado da imagem de entrada na etapa de segmentação, computando as equações 3 e 4. Cada k-ésimo elemento do histograma, $p_k(m)$, é computado paralelamente por uma arquitetura igual a apresentada na Fig. 4. Para isso, há L submódulos com arquiteturas iguais a essa nesse módulo. Conforme detalhado na Fig. 4, cada arquitetura do k-ésimo submódulo do histograma é formado por um comparador ($COMP_k$), um somador (SUM_k), dois registradores (R) e duas constantes (k e C).

Inicialmente, cada k-ésimo comparador verifica se o n-ésimo pixel de entrada, $a_{ij}(n)$, tem valor igual a k. Cada $COMP_k$ computa uma saída $c_{i,j}$, conforme a Equação 6. Essa saída é um valor booleano, quando igual a 1 ela habilita o k-ésimo somador. Toda vez que SUM_k é habilitado realiza-se a soma da constante C à saída anterior desse somador, funcionando assim como um acumulador. O valor de C é igual a $1/(M \times N)$. Após a entrada de todos os pixels da imagem, NHM apresenta como saída todos os componentes do

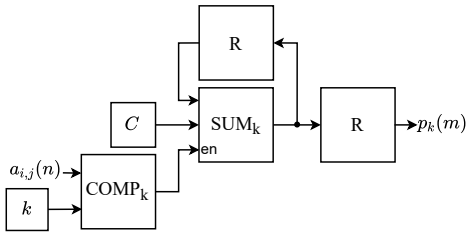


Figure 4: Arquitetura do k-ésimo submódulo do histograma normalizado em NHM.

histograma normalizado da m-ésima imagem, conforme a Equação 3.

As constantes k assumem em cada submódulo um valor de nível de cinza entre 0 e $L-1$, sendo assim representadas como um número inteiro sem sinal de 8 bits. Já a constante C, que possui um valor fracionário menor que 1, utiliza em sua representação apenas 24 bits na parte fracionária. Os somadores desse módulo, SUM_k, tem a saída com a mesma resolução de bits da constante C, visto que, os componentes do histograma normalizado assumem valores entre 0 e 1. Cada k-ésimo componente do histograma é transmitido para o PCOM e MIM.

4.2.3 Mean Intensity Module (MIM). O MIM realiza o cálculo da intensidade média dos pixels até o nível k, conforme a Equação 10. Cada k-ésima intensidade média, $\mu_k(m)$, é calculada paralelamente. A arquitetura desse módulo é apresentada na Fig. 5, ela é formada por 256 ganhos ($G_0, ..., G_{255}$), 1024 somadores ($SUM_1, ..., SUM_{1024}$) e 1280 registradores (R).

Baseado na Equação 10, cada k-ésimo componente do histograma normalizado é primeiramente multiplicado por um valor k, operação realizada no k-ésimo ganho, G_k, representado no diagrama de blocos da arquitetura. Posteriormente, cada k-ésimo valor de intensidade média, $\mu_k(m)$, é obtido através do somatório das saídas de todos os ganhos de índice 0 a k. A adição desses valores é realizada paralelamente com base no somador paralelo proposto por [23]. Através desse método há no máximo oito somadores em cascata para o cálculo de $\mu_k(m)$.

Todos os ganhos desse módulo, $G_0, ..., G_{255}$, têm sua saída representada como um número sem sinal de 8 bits na parte inteira e 24 na parte fracionária. Nos somadores, $SUM_1, ..., SUM_{1024}$, essa resolução de bits é mantida, visto que, a intensidade média dos pixels até um dado nível k poderá ser no máximo igual a 255, considerando como entrada da arquitetura geral uma imagem com o valor dos pixels representado por 8 bits. As $\mu_k(m)$ calculadas são fornecidas ao BCVM.

4.2.4 Probability of Class Occurrence Module (PCOM). O cálculo da probabilidade de ocorrência de classe para um limar k, $\omega_k(m)$, é realizado no PCOM com base na Equação 9. Todos os valores de $\omega_k(m)$ são computados paralelamente, através do somatório das k-ésimas entradas $p_k(m)$. Esse módulo possui uma arquitetura semelhante à do MIM, a única diferença entre eles é que as entradas não são multiplicadas por ganhos, sendo diretamente ligadas aos somadores $SUM_1, ..., SUM_{128}$. Assim, essa arquitetura é composta apenas por somadores e registradores, utilizando a mesma quantidade de componentes que o MIM.

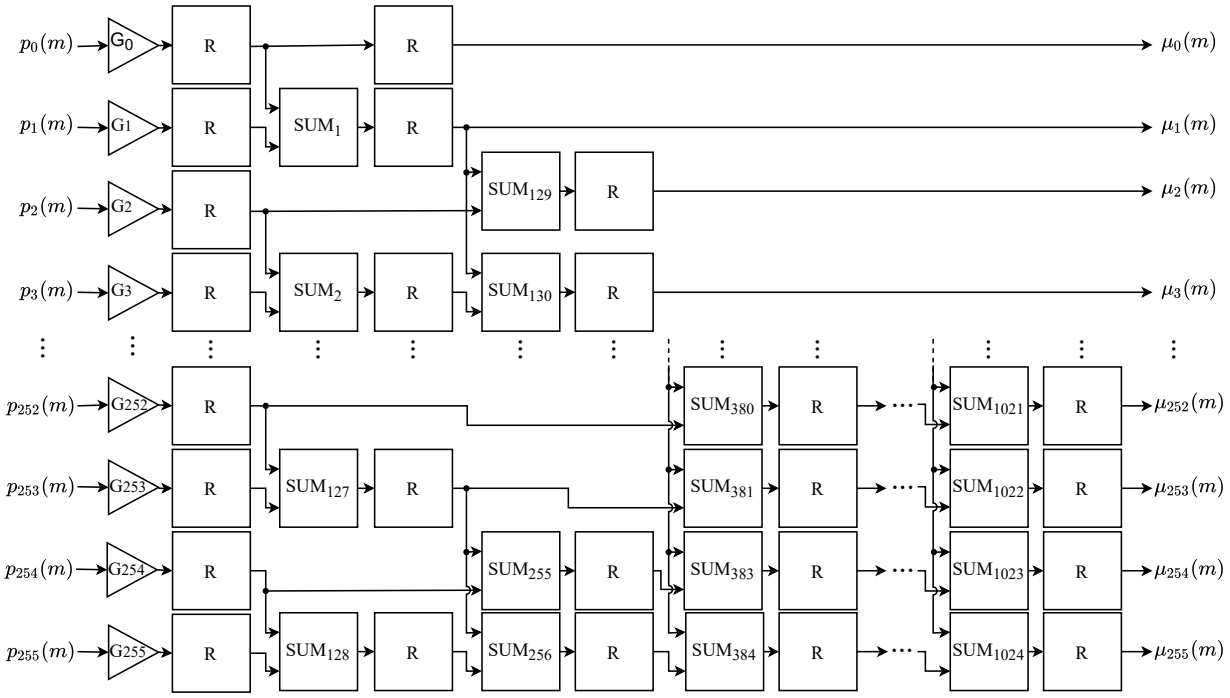


Figure 5: Arquitetura do *Mean Intensity Module*.

Os somadores dessa arquitetura, $SUM_1, ..., SUM_{1024}$, possuem uma resolução de bits diferentes dos empregados no MIM. Essas somas mantém a resolução de bits das entradas do módulo, $p_k(m)$, visto que, os valores da probabilidade de ocorrência de classe também podem está apenas entre 0 e 1. As probabilidades $\omega_k(m)$ calculadas são usadas como entrada do módulo BCVM.

4.2.5 Between-Class Variance Module (BCVM). As variâncias entre classes de cada m-ésima imagem, $\sigma_k^2(m)$, são calculadas pelo BCVM baseado-se na Equação 8. Cada k-ésima variância é computada paralelamente pela arquitetura apresentada na Fig. 6, havendo L submódulos iguais a esse no BCVM. Cada submódulo é composto por quatro multiplicadores ($MULT_{i,k}$), duas subtrações ($SUB_{i,k}$), um deslocamento de ponto (BPC_K), uma *Lookup Table* (LUT_k) e onze registradores (R).

O cálculo do numerador e denominador da divisão apresentada na Equação 8 são realizados em paralelo nessa arquitetura. O numerador é calculado efetuando primeiramente a multiplicação, $MULT_{1,k}$, da k-ésima probabilidade $\omega_k(m)$ pela média global da m-ésima imagem $\mu_{255}(m)$. O resultado dessa multiplicação é então subtraído por 1 pelo k-ésimo $SUB_{1,k}$ e, após isso, é multiplicado por ele mesmo pelo k-ésimo $MULT_{2,k}$. A saída desse multiplicador é o valor do denominador da divisão na Equação 8. Simultaneamente, o denominador é calculado realizando inicialmente a subtração da k-ésima probabilidade $\omega_k(m)$ do valor 1 pelo k-ésimo $SUB_{2,K}$. Esse resultado é então multiplicado pelo mesmo $\omega_k(m)$ na k-ésima $MULT_{3,k}$, sendo o valor obtido igual ao denominador da divisão.

A divisão dos valores calculados através da implementação direta desse operador aritmético é extremamente custoso, sendo essa operação o gargalo da arquitetura. Uma maneira de evitar o uso do divisor é realizar esse cálculo através da multiplicação do numerador pelo recíproco do denominador. Por definição, o recíproco de um número é o inverso dele, podendo esse valor ser aproximado através de uma LUT. Com isso, a divisão poderá ser calculada empregando apenas uma LUT e um multiplicador, proporcionando assim o aumento do *throughput*.

Assim, cada k-ésimo denominador calculado em $MULT_{3,k}$ tem seu recíproco aproximado pela k-ésima LUT_k. Essa LUT foi configurada com uma profundidade igual a 2^8, armazenando palavras de 33 bits, sendo 9 bits na parte inteira (com 1 bit para sinal) e 24 bits na parte fracionária. O mapeamento do valor de saída de cada k-ésima $MULT_{3,k}$ a um endereço da k-ésima LUT_k é realizado através do deslocamento do ponto binário oito bits à direita na k-ésima BPC_k. O valor aproximado do recíproco apresentado na saída de cada k-ésima LUT_k é então multiplicado no k-ésimo $MULT_{4,k}$ pela saída do k-ésimo $MULT_{2,k}$. Sendo o resultado dessa multiplicação a k-ésima variância entre classes da m-ésima imagem, $\sigma_k^2(m)$.

Todos os componentes desse módulo utilizam a resolução de 24 bits na parte fracionária, variando apenas o número de bits na parte inteira. A subtração $SUB_{1,k}$ emprega 9 bits na sua parte inteira, com 1 bit para sinal, enquanto $SUB_{2,k}$ não possui parte inteira, visto que, 1 menos $\omega_k(m)$ sempre será um valor positivo entre 0 e 1. O multiplicador $MULT_{1,k}$ utiliza 8 bits na parte inteira. $MULT_{2,k}$ emprega 18 bits, usando 1 bit para sinal. $MULT_{3,k}$ não possui parte inteira, visto que, suas duas entradas são valores entre 0 e 1. E $MULT_{4,k}$ usa 27 bits, com 1 bit para sinal. Cada k-ésima variância calculada é passada para o OTM.

4.2.6 Optimal Threshold Module (OTM). O último módulo da arquitetura geral é o OTM, que compara o valor das variâncias entre

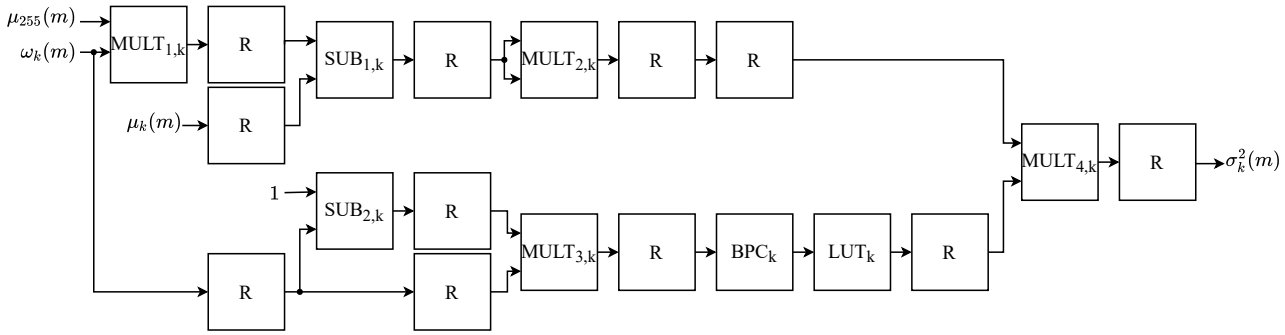


Figure 6: Arquitetura do k-ésimo submódulo da variância entre classes em BCVM.

classes, $\sigma_k^2(m)$, para determinar o limiar ótimo da m-ésima imagem, $k^*(m)$. A arquitetura desse módulo é apresentada na Fig. 7, ela é composta por 255 comparadores ($COMP_1$, ..., $COMP_{255}$), 510 multiplexadores (MUX_1, ..., MUX_{510}) e 765 registradores.

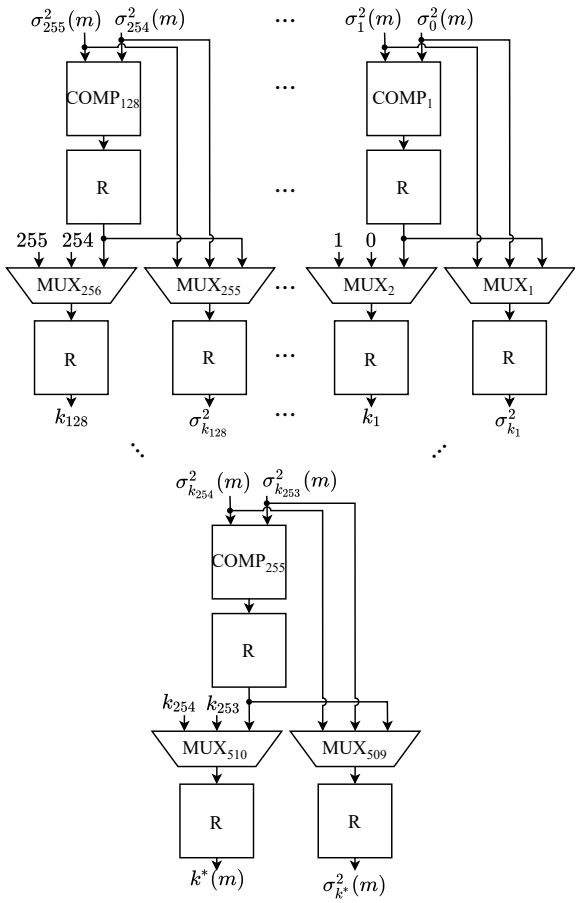


Figure 7: Arquitetura do *Optimal Threshold Module*.

Esse módulo implementa a Equação 7, buscando identificar o limiar k para qual existe o maior valor de variância entre classes. Para isso a comparação das variâncias da m-ésima imagem, $\sigma_k^2(m)$, é realizada através de uma árvore de comparadores. Cada k-ésimo componente $COMP_k$ compara se uma variância $\sigma_k^2(m)$ é maior que outra $\sigma_{k-1}^2(m)$. O resultado desse comparador é utilizado como seletor de dois multiplexadores, devendo definir como saída o valor da maior variância que foi comparada e o limiar k dessa variância. Todas as saídas dos multiplexadores são passadas para o próximo nível da árvore, até chegar ao último comparador, $COMP_{255}$, que irá selecionar como saída dos multiplexadores o valor do limiar ótimo da m-ésima imagem, $k^*(m)$, e sua variância entre classes, $\sigma_{k^*}^2(m)$.

5 RESULTADOS

5.1 Resultados da proposta de hardware

5.1.1 Resultados de ocupação do hardware. Através da síntese da arquitetura desenvolvida para o dispositivo FPGA alvo obteve-se um relatório de ocupação da área do hardware. A Tabela 1 apresenta os dados relacionados aos recursos utilizados por essa implementação. A primeira coluna apresenta o número de células lógicas ocupadas (NCL). A segunda exibe a quantidade de multiplicadores implementados usando blocos DSP (NMULT). E a terceira mostra o número de bits de blocos de memória utilizado (NBitsM).

Table 1: Recursos utilizados pelo sistema.

NCL	NMULT	NBitsM
591.946 (69%)	1.222 (80%)	2.162 K (4%)

Os dados apresentados na Tabela 1 evidenciam a viabilidade da implementação da arquitetura proposta no dispositivo FPGA alvo. Verifica-se que foi ocupado apenas 4% dos bits de blocos de memória disponíveis. Enquanto a quantidade de células lógicas ocupou 69%. O elemento mais utilizado foram os multiplicadores, utilizando 80% dos DSP do FPGA alvo. Essa implementação apresenta uma alta taxa de ocupação do FPGA alvo, porém isso é previsto devido ao alto grau de paralelismo da arquitetura.

5.1.2 Resultados do tempo de processamento. A Tabela 2 apresenta informações à respeito do tempo de processamento da implementação. A primeira coluna indica o *clock*, clk, em que opera o circuito. A segunda coluna apresenta o tempo de processar uma imagem, t_I. A terceira coluna exibe a latência inicial do sistema, D. E a quarta coluna mostra o *throughput* da arquitetura, th, que neste trabalho consiste na quantidade de imagens processadas por segundo. Conforme a Equação 2 o tempo de processamento de uma imagem

depende do seu tamanho. Assim, os dados apresentados na Tabela 2 consideram uma imagem de entrada de 250×375, resolução do fragmento extraído da imagem microscópica apresentada no item D da Fig. 2.

Table 2: Tempo de processamento do sistema.

Clk (ns)	t_I (ms)	D (ms)	th (IPS)
13,3	1,25	3,75	800

Os dados apresentados na Tabela 2 trazem informações importantes para a análise da aplicabilidade dessa arquitetura em tempo real. O *clock* de $13,3ns$, também corresponde ao valor do tempo de amostragem, t_s, visto que, todo o circuito opera com o mesmo *clock*. O tempo de processamento de uma imagem, t_I, com a resolução de 250×375 é igual a $1,25ms$, havendo uma latência inicial, D, de três vezes esse valor, sendo igual a $3,75ms$. Com isso, foi possível alcançar o *throughput* de 800 imagens processadas por segundo ou 75 Megapixels processados por segundo.

Através do *throughput* obtido, pode-se constatar a viabilidade do processamento em tempo real do vídeo capturado pelo sistema do Behavioral Genomics Lab. As imagens de alta resolução capturadas pelas câmeras desse sistema possuem aproximadamente 12 Megapixels, sendo assim possível processar cerca de 6,25 imagens por segundo com essa quantidade de pixel. Havendo quatro processos paralelos com a mesma arquitetura proposta neste trabalho, seria possível processar 25 imagens por segundo. Isso seria possível utilizando outro dispositivo FPGA com mais recursos de hardware, havendo várias opções disponíveis no mercado.

5.2 Comparação com trabalhos da literatura

A partir dos resultados obtidos, foram realizadas comparações com outros trabalhos do estado da arte. Os resultados foram comparados apenas com artigos que apresentaram dados de ocupação do hardware e tempo de processamento.

5.2.1 Comparação de ocupação do hardware. A Tabela 3 apresenta os recursos de hardware utilizados pelos trabalhos na literatura e uma comparação desses dados com a implementação proposta. A primeira coluna indica a referência analisada. A segunda, terceira, quarta e quinta coluna apresentam, respectivamente, o tipo de FPGA empregada, o número de células lógicas, número de multiplicadores e número de bits de blocos de memória da referência. A sexta, sétima e oitava coluna apresentam os mesmos dados de ocupação relacionados à arquitetura proposta. Por fim, as três últimas colunas apresentam a razão entre os componentes de hardware utilizados nessa proposta e os trabalhos da literatura.

O trabalho apresentado por [20] usou um dispositivo FPGA Xilinx Spartan 6 LX45 e teve uma ocupação de hardware de cerca de 1859 LUTs e 44 blocos de RAM. Esse FPGA usa cerca de 1,6 células lógicas (CL) por LUT, havendo utilizado cerca de 2975 CL, e cada bloco de RAM possui 18 Kbits, sendo assim usado 792 Kbits de memória. O número de multiplicadores utilizados foi não apresentado (NA).

Em [18] foi apresentado resultados para duas implementações distintas do método de Otsu, tendo como FPGA alvo a Altera Cyclone IV EP4CE115F29C6N. A comparação com esse trabalho será realizada considerado os melhores resultados apresentados nele, os quais foram obtidos com a implementação da técnica utilizando funções logarítmicas. O hardware implementado utilizou 2440 CL, 10,9 Kbits de memória e 46 multiplicadores.

O hardware apresentado em [19] utiliza o dispositivo FPGA Xilinx Virtex-5 xc5vfx70tffg1-136-1, havendo a ocupação de 161 *slices*, 4 BRAMs e 5 multiplicadores. Esse FPGA utiliza cerca de 6,4 CL por *slice* e cada bloco de memória possui 36 Kbits. Assim, foram utilizados cerca de 1030 CL e 144 Kbits de memória.

Através da Tabela 3 constatamos que a arquitetura proposta apresentou uma ocupação de hardware maior que todos os outros trabalhos da literatura. Contudo, isso é algo esperado, visto que, a implementação desenvolvida apresenta um alto grau de paralelismo, necessitando assim da utilização de mais recursos de hardware.

5.2.2 Comparação do throughput*.* A Tabela 4 apresenta uma comparação do *throughput* desse trabalho com outros da literatura. São apresentados os dados da resolução da imagem (RI) e o *clock* (Clk) empregados por cada referência. O *throughput* da referência (th_{ref}) e do presente trabalho (th_{work}) são apresentados, assim como *speedup* obtido com o hardware desenvolvido.

[20] apresenta resultados do tempo de execução empregando um clock de 10 ns e uma imagem de entrada com dimensões de 280×265, com representação dos pixels usando 8 bits. O tempo de processamento de uma imagem com o hardware proposto foi apresentado como igual a $1,489$ ms, sendo assim possível alcançar um *throughput* de $671,59$ IPS.

Em [18] foram apresentados dados de tempo e latência considerando um *clock* de $8,72$ ns e o processamento de uma imagem de resolução 1280×1024, também com representação dos pixels com 8 bits. O tempo de processamento de uma imagem não é apresentado diretamente, sendo aqui calculado como o valor do *clock* adotado vezes a quantidade de *clocks* necessários para a entrada de uma imagem e a obtenção da sua respectiva saída. Dessa forma, o tempo calculado é igual a $(1280 \times 1024 \times 2 + 536) \times 8,72 \cdot 10^{-9} \approx 22,87$ ms por imagem, sendo assim alcançado um throughput de $43,72$ IPS.

A proposta apresentada por [19] exibe resultados de *throughput* para o processamento de uma imagem de 640×480, com pixels representados por 8 bits, adotando um *clock* de $39,72$ ns. O valor de *throughput* apresentado indica a quantidade de mega bits processados por segundo (Mbps), sendo esse valor igual a $40,28$ Mbps. Como cada pixel possui 8 bits, temos que a quantidade de imagens processadas por segundo é obtido por $\frac{40,28 \cdot 10^6}{8 \times 640 \times 480} \approx 16,39$ IPS.

Os valores de th_{work} apresentados na Tabela 4 são calculados para cada referência considerando a adoção do mesmo *clock* e tamanho de imagem. O *throughput* é calculado como sendo a quantidade de pixels da imagem vezes o valor do *clock*, conforme Equação 2. O *speedup* obtido é igual à razão entre th_{work} e th_{ref}.

Os resultados apresentados na Tabela 4 demonstram que o alto grau de paralelismo da arquitetura desenvolvida proporcionou o alcance de significante *speedup* em relação aos outros trabalhos da literatura. A presente proposta obteve um tempo de processamento até 5 vezes mais rápido. O *speedup* obtido poderia ser ainda maior com a divisão da imagem em partes menores para o processamento dessas partes em paralelo.

Table 3: Comparação da ocupação do hardware com trabalhos da literatura.

Ref.	FPGA	NCL_{ref}	$NMULT_{ref}$	$NBitsM_{ref}$	NCL_{work}	$NMULT_{work}$	$NBitsM_{work}$	$R_{ocupacão}$ NCL	NMULT	NBitsM
[20]	Spartan 6	2975	NA	792 K	591.946	1.222	2.162 K	$\approx 198,97\times$	-	$\approx 2,73\times$
[18]	Cyclone IV	2440	46	10,94 K	591.946	1.222	2.162 K	$\approx 242,60\times$	$\approx 26,56\times$	$\approx 197,62\times$
[19]	Virtex 5	1031	5	144 K	591.946	1.222	2.162 K	$\approx 574,15\times$	$\approx 244,40\times$	$\approx 15,01\times$

Table 4: Comparação do *throughput* do hardware com trabalhos da literatura.

Ref.	RI	Clk (ns)	th_{ref} (IPS)	th_{work} (IPS)	*Speedup*
[20]	280×265	10,00	671,59	1351,35	$\approx 2,01\times$
[18]	1280×1024	8,72	43,72	87,49	$\approx 2,00\times$
[19]	640×480	39,72	16,39	81,97	$\approx 5,00\times$

6 CONCLUSÕES

Este trabalho teve como objetivo a implementação em hardware do método de Otsu, para utilização na segmentação de vermes em tempo real no projeto PHENOSPACE. O hardware foi desenvolvido em RTL utilizando ponto fixo, com uma implementação totalmente paralela. Foram apresentados todos os detalhes da arquitetura e os resultados de síntese referentes à ocupação de hardware e tempo de processamento, tendo como FPGA alvo a Intel Arria 10 GX 1150. Além disso, foram apresentados resultados de comparação em relação ao tempo de processamento e recursos de hardware utilizados com outras propostas da literatura. Conforme os dados apresentados, observou-se que a implementação obteve um *throughput* elevado, viabilizando a aplicação da arquitetura proposta na segmentação em tempo real dos vermes.

ACKNOWLEDGMENTS

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

REFERENCES

[1] Dajung Lee, Nirja Mehta, Alexandria Shearer, and Ryan Kastner. A hardware accelerated system for high throughput cellular image analysis. *Journal of Parallel and Distributed Computing*, 113:167–178, 2018. ISSN 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2017.11.013.

[2] Nao Nitta *et al.* Raman image-activated cell sorting. *Nature Communications*, 11 (1):3452, Jul 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-17285-3.

[3] Thomas Blasi *et al.* Label-free cell cycle analysis for high-throughput imaging flow cytometry. *Nature Communications*, 7(1):10256, Jan 2016. ISSN 2041-1723. doi: 10.1038/ncomms10256.

[4] Keisuke Goda *et al.* High-throughput single-microparticle imaging flow analyzer. *Proceedings of the National Academy of Sciences*, 109(29):11630–11635, 2012. ISSN 0027-8424. doi: 10.1073/pnas.1204718109.

[5] Keisuke Goda, Dino D. Carlo, and Bahram Jalali. Ultrafast automated image cytometry for cancer detection. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 129–132, 2013.

[6] Amir HajiRassouliha, Andrew J. Taberner, Martyn P. Nash, and Poul M. F. Nielsen. Suitability of recent hardware accelerators (dsps, fpgas, and gpus) for computer vision and image processing algorithms. *Signal Processing: Image Communication*, 68:101 – 119, 2018. doi: https://doi.org/10.1016/j.image.2018.07.007.

[7] Frank Vahid. *Digital Design with RTL Design, Verilog and VHDL*. John Wiley & Sons, Massachusetts, 2 edition, 2010.

[8] PHENOSPACE. Quantifying behavioural phenotype space: chemistry-to-gene screens and combination therapies, 2020. URL https://cordis.europa.eu/project/id/714853/reporting.

[9] Wei Wang and Xinming Huang. An FPGA co-processor for adaptive lane departure warning system. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 1380–1383, May 2013. doi: 10.1109/ISCAS.2013.6572112.

[10] Jin Zhao, Bingqian Xie, and Xinming Huang. Real-time lane departure and front collision warning system on an FPGA. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–5, Sep. 2014. doi: 10.1109/HPEC.2014.7041003.

[11] Ali Alhamwi, Bertrand Vandeportaele, and Jonathan Piat. Real Time Vision System for Obstacle Detection and Localization on FPGA. In *Computer Vision Systems*, pages 80–90, Cham, 2015. Springer International Publishing. ISBN 978-3-319-20904-3.

[12] Xin Ren and Yu Wang. Design of a FPGA hardware architecture to detect real-time moving objects using the background subtraction algorithm. In *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*, pages 428–433, Dec 2016. doi: 10.1109/ICCSNT.2016.8070194.

[13] Chetan Tulasigeri and M. Irulappan. An advanced thresholding algorithm for diagnosis of glaucoma in fundus images. In *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 1676–1680, May 2016. doi: 10.1109/RTEICT.2016.7808118.

[14] Hi S. Kim. FPGA-based of thermogram enhancement algorithm for non-destructive thermal characterization. *International Journal of Engineering*, 31(10): 1675–1681, 2018. ISSN 1025-2495.

[15] Wang Jianlai, Yang Chunling, Zhu Min, and Wang Changhui. Implementation of Otsu's thresholding process based on FPGA. In *2009 4th IEEE Conference on Industrial Electronics and Applications*, pages 479–483, May 2009. doi: 10.1109/ICIEA.2009.5138252.

[16] Hui Tian, Siew-Kei Lam, and Thambipillai Srikanthan. Implementing Otsu's thresholding process using area-time efficient logarithmic approximation unit. In *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, volume 4, pages IV–IV, May 2003. doi: 10.1109/ISCAS.2003.1205763.

[17] Jai G. Pandey, Abhijit Karmakar, Chandra Shekhar, and S. Gurunarayanan. A Novel Architecture for FPGA Implementation of Otsu's Global Automatic Image Thresholding Algorithm. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 300–305, Jan 2014. doi: 10.1109/VLSID.2014.58.

[18] Albert F. Torres-Monsalve and Jaime Velasco-Medina. Hardware implementation of ISODATA and Otsu thresholding algorithms. In *2016 XXI Symposium on Signal Processing, Images and Artificial Vision (STSIVA)*, pages 1–5, Aug 2016. doi: 10.1109/STSIVA.2016.7743329.

[19] Jai G. Pandey and Abhijit Karmakar. Unsupervised image thresholding: hardware architecture and its usage for FPGA-SoC platform. *International Journal of Electronics*, 106(3):455–476, 2019. doi: 10.1080/00207217.2018.1540065.

[20] Rahul K. Das, Abhishek De, Chandrajit Pal, and Amlan Chakrabarti. DSP hardware design for fingerprint binarization and thinning on FPGA. In *Proceedings of The 2014 International Conference on Control, Instrumentation, Energy and Communication (CIEC)*, pages 544–549, Jan 2014. doi: 10.1109/CIEC.2014.6959148.

[21] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, Upper Saddle River, 3 edition, 2009.

[22] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, Jan 1979. ISSN 0018-9472. doi: 10.1109/TSMC.1979.4310076.

[23] Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. *J. ACM*, 27(4):831–838, October 1980. ISSN 0004-5411. doi: 10.1145/322217.322232.