Hardware Implementation of the GVF Approach for Deformable Contour Detection on FPGA

Mohamed Lamine Hamidatou
National Higher School of Agronomy (ENSA), Algiers, Algeria | Systems Design Methods Laboratory, National High College of Data Processing, Algiers, Algeria
med-lamine.hamidatou@edu.ensa.dz (corresponding author)

Fatma Zohra Hamadi
Department of Electronics, Polytechnic National College. Algiers, Algeria
Fatma.hamidatou@g.enp.edu.dz

Latifa Hamami
Department of Electronics, Polytechnic National College. Algiers, Algeria
latifa.hamami@g.enp.edu.dz

Bouchra Bouzid
Department of Electronics, Polytechnic National College, Algiers, Algeria
bouchrabouzid120@gmail.com

Sabrina Ait Belkacem
Department of Electronics, Polytechnic National College, Algiers, Algeria
sabrina.ait_belkacem@g.enp.edu.dz

Fatima Zohra Allam
Department of Electronics, Polytechnic National College, Algiers, Algeria
fatima_zohra.allam@g.enp.edu.dz

Received: 13 September 2025 | Revised: 13 October 2025 | Accepted: 19 October 2025

 DOI: https://doi.org/10.48084/etasr.14770

ABSTRACT

Active contours, or snakes, are widely used in medical image segmentation due to their ability to accurately delineate object boundaries. The Gradient Vector Flow (GVF) model enhances traditional snakes by improving convergence and effectively capturing concave shapes. This paper presents a hardware implementation of the GVF algorithm on a PYNQ-Z2 FPGA using a modular architecture designed to optimize computation and parallelism. The algorithm was first developed and validated in MATLAB to evaluate its accuracy and stability on medical images. This step enabled fine-tuning of key GVF parameters, such as the regularization coefficient and the number of iterations, to ensure reliable convergence and precise contour segmentation. It was then implemented in Vivado HLS and translated into an optimized hardware architecture that leverages FPGA parallelism and pipelining to minimize latency and enhance performance. Experimental results demonstrate real-time operation, high segmentation accuracy, and low latency, confirming the suitability of this approach for embedded medical imaging applications requiring both speed and precision.

Keywords-medical image segmentation; snakes; Gradient Vector Flow (GVF); FPGA; MATLAB simulation

I. INTRODUCTION

Active contour-based segmentation consists of dividing an image into homogeneous and connected regions using an evolving curve, whose position is guided by a specific discretization criterion. This approach has been extensively studied for image segmentation.

For instance, authors in [1] proposed a fully automatic method for Left Ventricle (LV) segmentation on Magnetic Resonance (MR) images, based on an implicit deformable model (level set) in a variational framework that combines edge-, region-, and shape-based information. Another example is a semi-automatic approach for brain tumor segmentation, which combines the Gradient Vector Flow (GVF) snake model (introduced by authors in [2] and extended in [3]) with the Canny edge detector [4]. The practical application of this hybrid method to brain Magnetic Resonance Imaging (MRI) images has shown effective segmentation performance [5].

The GVF method improves the performance of classical active contours, particularly in terms of initialization and convergence within concave regions. It has become a reference technique for accurate segmentation in complex contexts.

In this work, we present a modular FPGA-based implementation of the GVF model, aiming to achieve real-time performance and low energy consumption for medical image segmentation. Our contribution demonstrates how a hardware-oriented design can accelerate GVF computation while maintaining segmentation accuracy.

II. GRADIENT VECTOR FLOW

The computation of the GVF field is performed in two main steps.

A. Edge Map Computation

The edge map $f(x,y)$ represents the derivative of the image, generally expressed as in (1):

$$f(x,y) = \left|\nabla\left(G_\sigma(x,y) * I(x,y)\right)\right|^2 \quad (1)$$

This operation generates vectors pointing toward the image edges.

B. External Force Field

The GVF, introduced by authors in [2] and extended in [3], is a vector field defined over the entire image domain. It attracts active contours toward distant edges by minimizing an energy functional that combines spatial smoothness with fidelity to the gradient:

$$E = \iint \left[\mu\left(u_x^2 + u_y^2 + v_x^2 + v_y^2\right) + |\nabla f|^2 * |V - \nabla f|^2\right] dxdy \quad (2)$$

where:

$$u_x = \frac{\partial u}{\partial x}, u_y = \frac{\partial u}{\partial y}, v_x = \frac{\partial v}{\partial x}, v_y = \frac{\partial v}{\partial y},$$

and:

$$V = \left(u(x,y), v(x,y)\right)$$

is the GVF field to be computed, ∇f is the gradient of the (typically smoothed) image, and μ is a regularization parameter that balances fidelity to the gradient and the smoothness of the field.

The GVF is obtained by solving the equations derived from the Euler–Lagrange formulation, given as (3) and (4):

$$\mu\nabla^2 u - \left(f_x^2 + f_y^2\right) * (u - f_x) = 0 \quad (3)$$

$$\mu\nabla^2 v - \left(f_x^2 + f_y^2\right) * \left(v - f_y\right) = 0 \quad (4)$$

where f_x and f_y are the partial derivatives of f with respect to x and y, respectively.

We consider u and v as time-dependent functions, since the contour evolves over time t. This leads to (5) and (6), known as the generalized diffusion equations:

$$u_t(x,y,t) = \mu\nabla^2 u(x,y,t) - [u(x,y,t) - f_x(x,y)] * \left[f_x(x,y)^2 + f_y(x,y)^2\right] \quad (5)$$

$$v_t(x,y,t) = \mu\nabla^2 v(x,y,t) - \left[v(x,y,t) - f_y(x,y)\right] * \left[f_x(x,y)^2 + f_y(x,y)^2\right] \quad (6)$$

with:

$$\begin{cases} b_n(x,y) = f_x(x,y)^2 + f_y(x,y)^2 \\ C_1(x,y) = b(x,y) f_x(x,y) \\ C_2(x,y) = b(x,y) f_y(x,y) \end{cases} \quad (7)$$

Equations (5), (6), and (7) were discretized in [2, 5], enabling a numerical implementation of the model. The energy functional associated with these equations is designed to optimize the GVF field in order to improve the convergence of active contours, as described in [2, 3] and later generalized in [6].

III. DESCRIPTION OF THE GRADIENT VECTOR FLOW ALGORITHM

Figure 1 shows a flowchart summarizing the main steps of the GVF segmentation method, from image preprocessing to obtaining the segmented result.

The active contour initialization begins with a circle centered on the image, with its center and radius adjusted according to the image type (synthetic or real). Before applying the GVF method for segmentation, preprocessing is essential, as medical images may contain noise and artifacts that interfere with contour detection. Anisotropic filtering, based on directed diffusion, is used to reduce noise while preserving important edges and structures, thus ensuring better segmentation [7].

After preprocessing, the contour map is generated by applying a Gaussian–Laplacian filter, followed by squaring to accentuate gradient transitions. External forces, derived from gradients using Sobel masks, guide the deformation of the active contour. To ensure stable propagation, these force fields are regularized using a four-connectivity Laplacian filter.

The deformation of the active contour is based on the calculation of first-, second-, and fourth-order derivatives of the contour points. At each iteration, the coordinates are updated according to a balance between internal forces (continuity,

curvature) and external forces (image). Interpolation follows each update to distribute the points evenly, ensuring regularity, numerical stability, and improved segmentation accuracy.

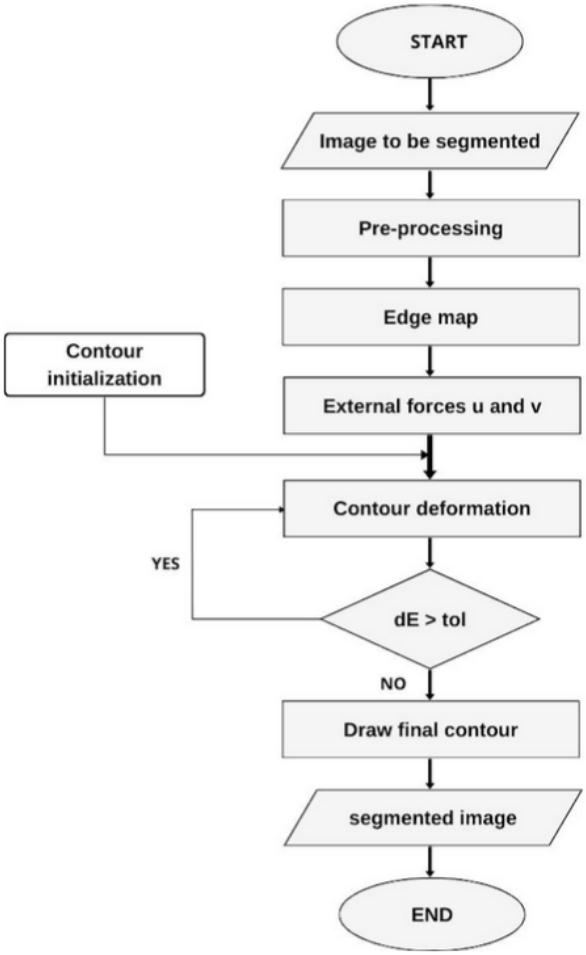


Fig. 1. Flowchart illustrating the main steps of the GVF algorithm.

IV. HARDWARE IMPLEMENTATION OF THE GVF ALGORITHM.

Our implementation relies on the PYNQ-Z2 board, which integrates a Xilinx Zynq-7000 SoC. This device combines a dual-core ARM Cortex-A9 processor (Processing System (PS)) with FPGA-based Programmable Logic (PL) [8, 9], Such an architecture enables efficient hardware/software co-design, making it well-suited for accelerating complex algorithms like GVF.

In this section, we present the hardware implementation strategy adopted for the Zynq platform, where the GVF algorithm was modularly divided into three hardware-accelerated stages.

A. Modular Implementation of the GVF Algorithm

Due to the complexity and resource demands of the GVF algorithm, a modular implementation strategy was adopted. The algorithm was split into three stages, each implemented and tested independently.

- Edge map generation (Part 1): This first module extracts edge information from a preprocessed image. It computes horizontal and vertical gradients (f_x, f_y), the gradient magnitude (b_n), and curvature measures (C_1, C_2) to identify intensity discontinuities corresponding to object boundaries.
- Part 2: External force fields (Part 2): The second module computes the external vector fields (u, v) from the parameters (f_x, f_y, b_n, C_1, C_2) obtained in the first stage. These fields guide the contour toward object boundaries.
- Contour deformation (Part 3): This module takes (u, v) and initial contour coordinates (x_i, y_i) to compute the final contour (x_f, y_f), based on the balance of internal and external forces.

B. Implementation Steps of the GVF Algorithm on FPGA

The implementation steps are described below and illustrated in Figure 2.

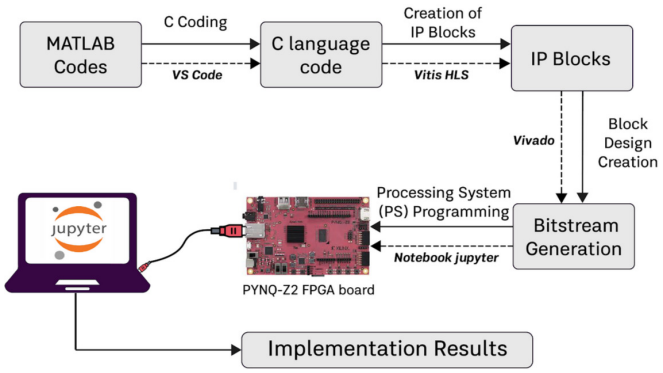


Fig. 2. Simplified schematic of the HLS-based design flow.

1) Coding in C

The GVF algorithm is first coded in C, defining the necessary computations and operations to simulate the vector field from the input image. This coding phase is carried out while considering performance constraints related to the FPGA architecture and parallelization of calculations. The goal is to optimize the code so that it can be easily converted into hardware blocks using Vitis HLS.

2) Creation of Intellectual Property Blocks

The prepared C code is imported into Vitis HLS, a high-level synthesis environment. Vitis HLS translates the code into an optimized hardware description in HDL, generating Intellectual Property (IP) blocks that implement the algorithm as parallel hardware components, thereby enhancing performance. These blocks are independent, reusable modules that can be integrated into the block design in Vivado. The overall design and implementation flow were carried out using Vitis HLS for high-level synthesis [10].

3) Generation of the HDL Wrapper

After the IP blocks are created, the next step is to generate the HDL Wrapper for each IP block. The wrapper is essential because it encapsulates each IP block into a structure

compatible with the FPGA architecture. The wrapper handles the interfaces between the IP blocks and the control signals needed for communication with other system components. This process is carried out in Vivado, ensuring the correct integration of these blocks into the block design.

4) Block Design Creation and Bitstream Generation

In Vivado, the creation of block designs starts by establishing the required connections between the system's master and slave blocks [11]. The Vivado Design Suite further supports hardware integration, synthesis, and bitstream generation, offering a comprehensive environment for optimized IP core generation and system-level integration. Memory address configuration is then carried out using the Address Editor tool, which allows the adjustment of the allocated addresses for each component. Once the architecture is validated, the next step is to generate the HDL wrapper, which encapsulates the block design into a synthesizable hardware description. This description is then subjected to synthesis in Vivado, a process where the Register Transfer Level (RTL) description is translated into a hardware structure composed of logical elements such as Look-Up Tables (LUTs), registers, flipflops, and memory blocks.

An illustration of this block design for Part 1 is presented in Figure 3, showing how the IP blocks are integrated and interconnected within the system architecture. FPGA design optimization includes eliminating redundant operations in combinatorial logic, synchronizing signals to reduce cycles in sequential logic, and efficiently allocating hardware resources. These adjustments aim to improve the system's performance. Synthesis reports, particularly on energy consumption, are generated to evaluate the energy efficiency of the design.

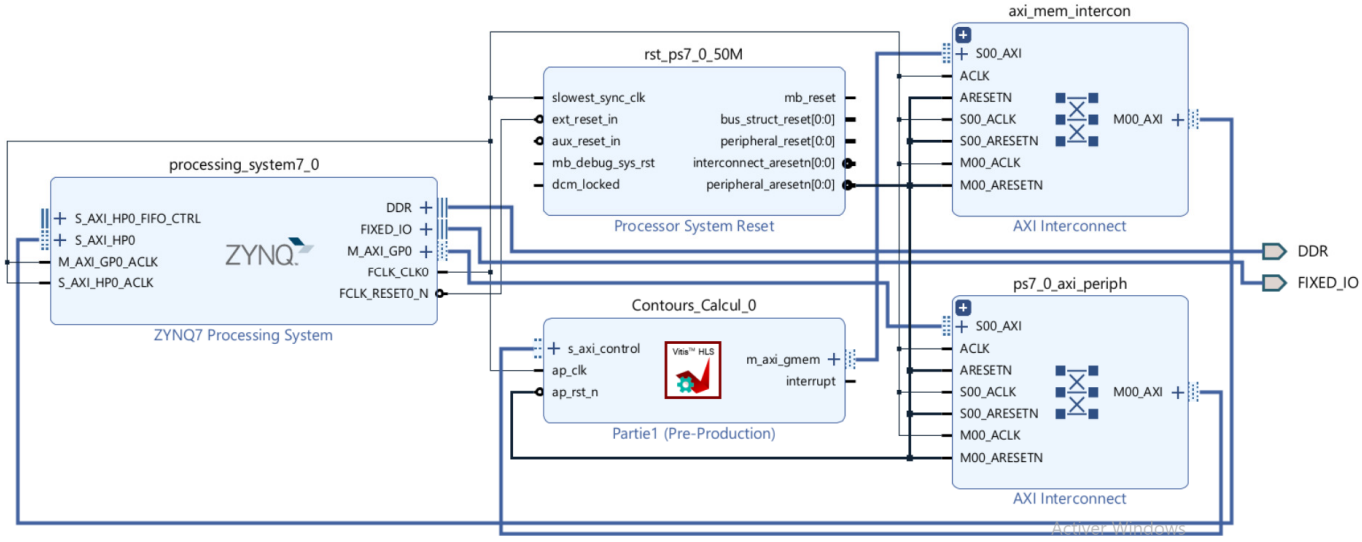


Fig. 3. Block design for Part 1: Edge map generation.

The final step in the Vivado design flow is the system implementation and generation of the bitstream file, transforming the synthesized design into a physical configuration executable on the FPGA. The placement of logical components and routing establishes the physical connections between them, whereas clock and I/O planning ensure synchronization and signal assignment to the pins.

Timing analysis and design rule verification guarantee the adherence to timing constraints and technological compliance. Once these steps are validated, Vivado generates the bitstream, a binary file for FPGA programming. Two essential files are generated: a TCL file to automate the design reconstruction, and an HWH file containing metadata necessary for interaction with the host processor.

5) Programming the Processing System

The programming of the CPU–FPGA system is performed in Python within Jupyter Notebook, using essential files to leverage IP blocks as Overlays, allowing interaction between the PL and PS. After programming the first part of the algorithm (contour map) via Jupyter Notebook, the execution of the remaining hardware blocks in the GVF architecture follows the same methodology. The results from the first execution, exported as .csv files (f_x, f_y, b_n, C_1, C_2), are used as inputs for the second hardware block, the external force field calculation (GVF_Calcul). The corresponding bitstream is loaded into the FPGA, and a new Overlay is initialized in Jupyter Notebook. Memory buffers are allocated for each input (f_x, f_y, b_n, C_1, C_2) and the expected output (u, v) using the allocate() function, and the input files are loaded into the buffers. A start signal is sent to the IP block to execute the GVF calculation, measured using the time library, and the results (u, v) are validated and saved as .csv files.

These results serve as inputs for the active contour deformation, where the previous steps are repeated to extract the final coordinates of the deformed contour. The hardware implementation is then validated by comparing the outputs with MATLAB results, following a methodology similar to other FPGA-based image processing implementations [12].

C. Resource Utilization and Power Consumption

Resource utilization and power consumption were evaluated using the Vivado Design Suite reporting tools, providing detailed estimates of LUTs, Flip-Flops (FFs), Digital Signal Processors (DSPs), Block RAMs (BRAMs), and both static and dynamic power [13]. Figures 4 and 5 illustrate the results, detailing hardware resource utilization and energy consumption. Similar evaluations have been reported in other FPGA-based image processing and segmentation implementations [14], confirming the relevance of such analyses for assessing hardware performance and efficiency.

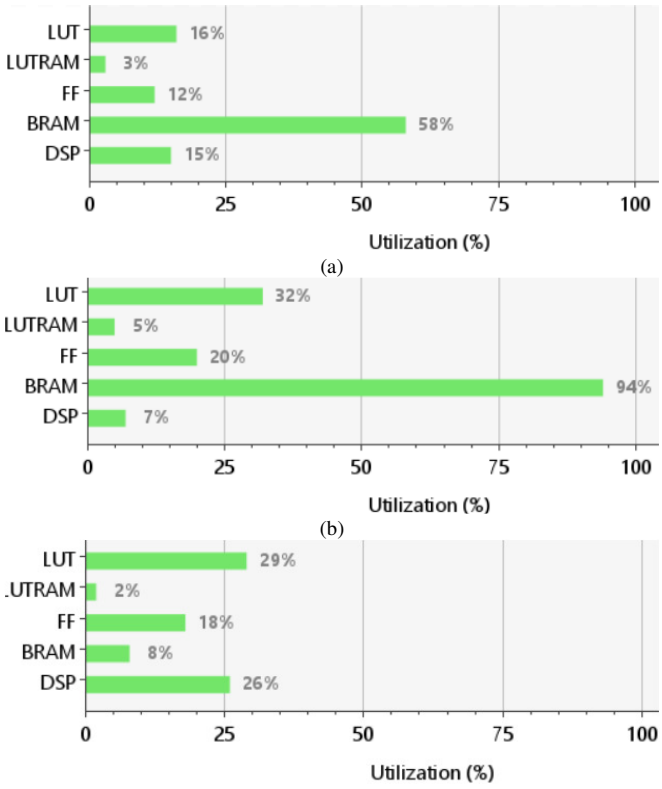
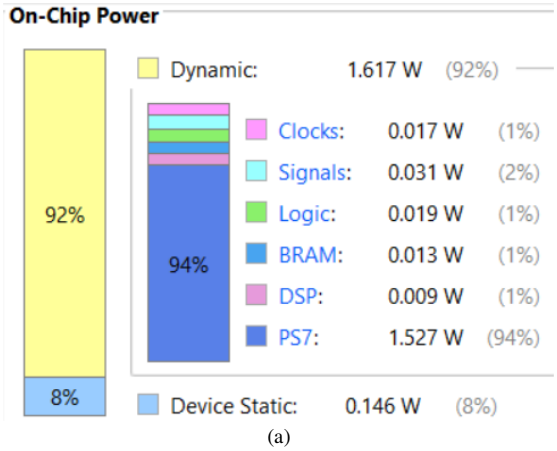


Fig. 4. Summary usage report of GVF algorithm: (a) Part 1: Edge map generation, (b) Part 2: External force fields, (c) Part 3: Contour deformation.



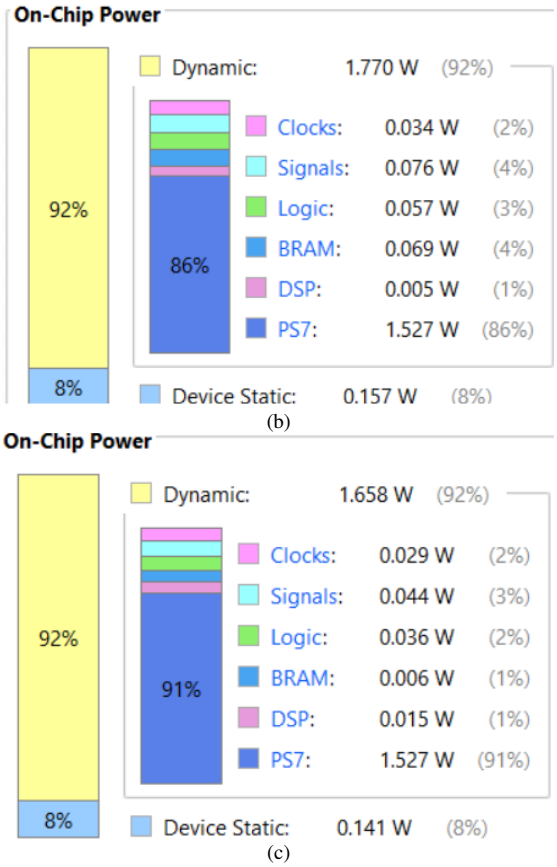


Fig. 5. Power report for the GVF algorithm: (a) Part 1: Edge map generation, (b) Part 2: External force fields, (c) Part 3: Contour deformation.

To clarify, LUTs and FFs represent the logic resources used to implement computational and control operations within the FPGA. DSPs and BRAMs indicate the arithmetic and memory resources employed to accelerate matrix and convolution operations.

V. RESULTS AND DISCUSSION

A. GVF Results Simulated in MATLAB for Brain MRI Images

Figure 6 illustrates the segmentation results of two brain MRI images obtained using the GVF-based active contour model. The images on the left show the initial MRI scans with manually drawn starting contours, whereas the images on the right present the evolution of the active contours leading to the final stabilized contours, accurately outlining the brain structures.

The implementation parameters were set as follows: $\mu = 0.15$, $\alpha = 0.1$, $\beta = 0.06$, $tol = 0.004$, with a maximum number of iterations limited to 300. The adjustment of these parameters strongly influences the performance of the GVF model: the μ parameter controls the smoothness of the vector field, whereas α and β define the balance between internal and external forces. The tolerance value (tol) determines the stability of the convergence.

Thus, an appropriate selection of these parameters (μ, α, β, and tol) leads to a more accurate and stable segmentation,

whereas improper tuning may cause contour leakage or insufficient boundary precision. Under this configuration, the active contour converges in a stable and robust manner, enabling precise, consistent, and reliable segmentation of the main brain structures, even in regions with intensity variations or weakly defined boundaries.

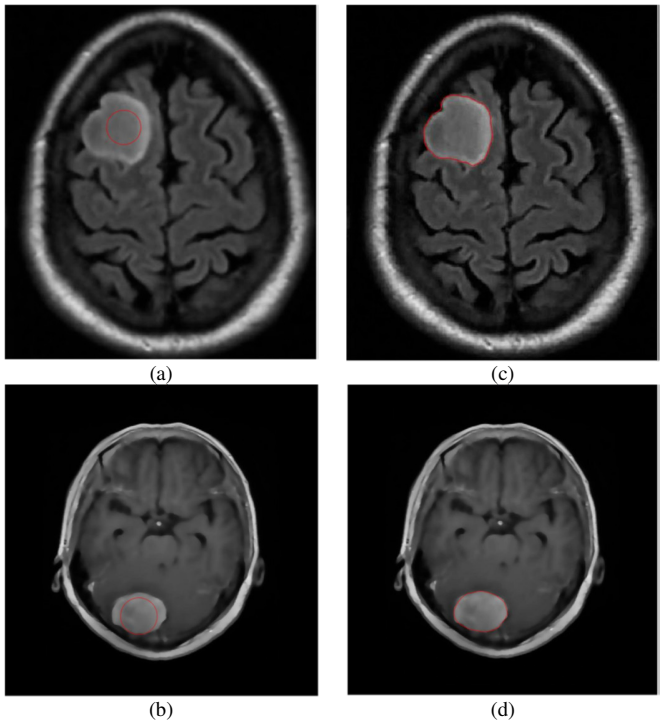


Fig. 6. Segmentation results simulated in MATLAB using GVF on two brain MRI images: (a, b) initial scans, (c, d) final stabilized contours.

B. GVF Model Implementation Results on FPGA

To display the segmentation results on the FPGA, the coordinates of the final contour were retrieved from the Jupyter Notebook and displayed in MATLAB.

Figure 7 presents the segmentation results obtained from the hardware implementation of the GVF-based active contour model on the FPGA platform, applied to brain MRI images. The contours generated by the FPGA closely match those produced by the MATLAB simulation, demonstrating the functional equivalence between the hardware and software implementations.

From a qualitative perspective, both results exhibit similar contour convergence and boundary smoothness, confirming that the FPGA implementation preserves the stability and precision of the GVF-based segmentation model. From a quantitative perspective, slight numerical discrepancies were observed, mainly due to the use of fixed-point arithmetic and hardware parallelization, which may slightly alter the precision of intermediate computations. However, these differences remain negligible and do not affect the overall segmentation accuracy or the delineation of fine anatomical structures.

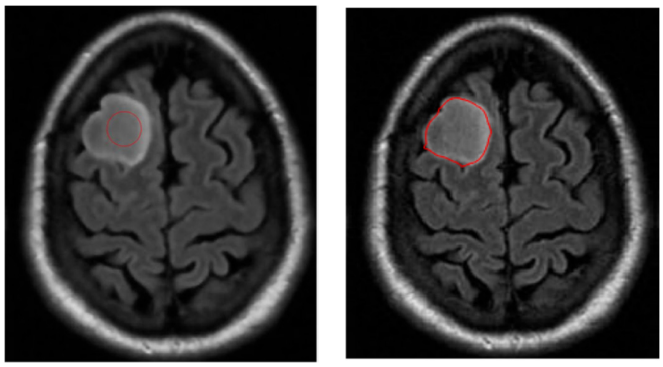


Fig. 7. Segmentation results using FPGA on a brain MRI image: (a) initial scan, (b) final stabilized contour.

This high level of consistency validates the reliability of the proposed FPGA architecture and highlights its capability to reproduce MATLAB-like results while significantly reducing computation time and power consumption. Overall, these results confirm the efficiency and robustness of the proposed hardware design, making it suitable for real-time medical image segmentation applications.

Table I summarizes hardware resource utilization, execution times, and power consumption for each processing module, providing a comparative analysis of the three implementation parts. Part 1 exhibits a balanced configuration with moderate resource usage and an average execution time, whereas Part 2 requires higher resource and power allocation. Part 3 demonstrates superior efficiency with a significantly reduced computation time. The computation time indicates the total execution latency of each hardware module. The static power represents idle power consumption, whereas dynamic power corresponds to the additional energy drawn during active processing. Overall, these results highlight the trade-offs between resource utilization, processing speed, and energy efficiency across the three FPGA design partitions.

TABLE I. FPGA RESOURCE UTILIZATION, EXECUTION TIME, AND POWER CONSUMPTION PER PART

Metric	Part 1	Part 2	Part 3
Resource utilization (%)	LUT: 16 LUTRAM: 3 FF: 12 DSP: 15 BRAM: 58	LUT: 32 LUTRAM: 5 FF: 20 DSP: 7 BRAM: 94	LUT: 29 LUTRAM: 2 FF: 18 DSP: 26 BRAM: 8
Computation time (s)	1.4380	8.4127	0.1746
Power consumption (W)	Static: 0.12 Dynamic: 0.78	Static: 0.15 Dynamic: 1.23	Static: 0.10 Dynamic: 0.45

VI. CONCLUSION

The FPGA-based implementation of the Gradient Vector Flow (GVF) algorithm successfully leveraged the parallel processing capabilities of hardware to enhance performance in terms of latency, throughput, and resource efficiency. By modularly partitioning the algorithm into optimized Intellectual Property (IP) blocks using Vitis HLS and Vivado, each

computational stage was refined to balance execution speed, hardware utilization, including Block RAM (BRAM), Look-Up Tables (LUTs), Digital Signal Processors (DSPs), and Flip-Flops (FFs), and result accuracy. The close agreement between the FPGA and MATLAB outputs validates the effectiveness of the proposed approach.

Overall, this work provides a concrete step toward real-time, energy-efficient, and clinically relevant hardware acceleration of complex medical image segmentation algorithms.

ACKNOWLEDGMENT

The authors would like to express their deep gratitude to Prof. R. Benyahia from the Medical Imaging Department at CPMC Mustapha Bacha Hospital, Algiers, for his invaluable support in providing the medical images essential for this study.

REFERENCES

[1] C. Beitone, C. Tilmant, and F. Chausse, "Fully Automatic Deformable Model Integrating Edge, Texture and Shape - Application to Cardiac Images Segmentation," in *10th International Conference on Computer Vision Theory and Applications*, Berlin, Germany, 2025, pp. 517–522, https://doi.org/10.5220/0005304005170522.

[2] C. Xu and J. L. Prince, "Gradient vector flow: a new external force for snakes," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Juan, PR, USA, 1997, pp. 66–71, https://doi.org/10.1109/CVPR.1997.609299.

[3] C. Xu and J. L. Prince, "Snakes, shapes, and gradient vector flow," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 359–369, Mar. 1998, https://doi.org/10.1109/83.661186.

[4] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986, https://doi.org/10.1109/TPAMI.1986.4767851.

[5] W. Guoqiang and W. Dongxue, "Segmentation of Brain MRI Image with GVF Snake Model," in *2010 First International Conference on Pervasive Computing, Signal Processing and Applications*, Harbin, China, 2010, pp. 711–714, https://doi.org/10.1109/PCSPA.2010.177.

[6] C. Xu and J. L. Prince, "Generalized gradient vector flow external forces for active contours1," *Signal Processing*, vol. 71, no. 2, pp. 131–139, Dec. 1998, https://doi.org/10.1016/S0165-1684(98)00140-6.

[7] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 629–639, July 1990, https://doi.org/10.1109/34.56205.

[8] Xilinx, "PYNQ: Python productivity for Zynq." PYNQ. http://www.pynq.io/.

[9] Xilinx, *Zynq 7000 SoC Technical Reference Manual (UG585)*, (2023). [Online]. Available: https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM/Quad-SPI-Controller.

[10] Xilinx, *Vitis High-Level Synthesis User Guide (UG1399)*, (2025). Available: https://docs.amd.com/r/en-US/ug1399-vitis-hls.

[11] Xilinx, *Vivado Design Suite User Guide: Getting Started (UG910)*, (2025). Available: https://docs.amd.com/r/en-US/ug910-vivado-getting-started/Design-Hubs.

[12] A. Alhomoud *et al.*, "Model-based Design of a High-Throughput Canny Edge Detection Accelerator on Zynq-7000 FPGA," *Engineering, Technology & Applied Science Research*, vol. 14, no. 2, pp. 13547–13553, Apr. 2024, https://doi.org/10.48084/etasr.7081.

[13] Xilinx, *Vivado Design Suite User Guide: Power Analysis and Optimization (UG907)*, (2025). Accessed: Oct. 30, 2025. [Online]. Available: https://docs.amd.com/r/en-US/ug907-vivado-power-analysis-optimization.

[14] F. Z. Hamadi, M. L. Hamidatou, L. Hamami-Mitiche, S. A. Belkacem, and B. Bouzid, "MATLAB Simulation, and FPGA Implementation of the DRLSE Segmentation Algorithm," *Traitement du Signal*, vol. 42, no. 2, pp. 719–727, Apr. 2025, https://doi.org/10.18280/ts.420210.