Comparitive Analysis of Edge Detection Algorithm on HLS and HDL for MRI Images

Rithanathith Senthilkumar
Electrical and Computer Engineering
Georgia Institute of technology
rsenthilkumar9@gatech.edu

Saniya Amit Datir
Electrical and Computer Engineering
Georgia Institute of technology
sdatir3@gatech.edu

***Abstract*—Early detection of brain tumors through MRI is crucial for better survival rates, and the Canny algorithm is effective in accurately detecting lower edges. To address the limitations of conventional system designs, we propose a hardware architecture for the Canny algorithm using the Zynq 7000 series architecture, which offers hardware capability on SOC while enhancing software programmability. Our design includes an efficient data reordering module for mapping signals to the correct network, and utilizes DMA IP and custom IP for edge detection operations. In our work, we compare the performance of the HLS and HDL approaches for implementing the Canny algorithm on the Pynq-Z2 board, and analyze the causes for any differences in performance. This project evaluates the most efficient architecture for the Canny algorithm using both HLS and HDL approaches.**

***Index Terms*—Image Processing, MRI, FPGA, Canny, HLS**

I. Introduction

In digital image processing, we use techniques to detect points in images where the brightness changes, which we call edges. There are different methods for edge detection, such as Sobel, Prewitt, fuzzy logic, Roberts, and Canny edge detection. Among these, Canny edge detection is the most precise technique, which can detect edges of objects with low errors, high accuracy, and localization [1]. Edge detection is useful in many fields like medical imaging, fingerprint recognition, and satellite imagery.

The accurate detection of soft tissue and lesions in MRI images is crucial for medical diagnosis and treatment planning, and the Canny edge detection algorithm excels at this task [2]. Utilizing an FPGA for implementing this algorithm can result in significant performance gains over software-based approaches, thanks to the parallel processing capabilities of FPGAs. Additionally, the flexibility of FPGAs allows developers to customize the algorithm for their specific application and hardware platform. However, optimizing and designing an FPGA-based implementation of the Canny algorithm requires careful consideration of factors like memory usage, pipeline design, and clock speed. Overall, FPGA-based Canny edge detection implementations offer high-performance solutions for image processing applications, especially in medical fields where accuracy and speed are paramount.

II. Problem Description

Early detection of tumors leads to a better chance of survival Detection of brain tumors MRI images. MRI images are commonly used for brain tumor detection, but other algorithms may not accurately detect lower edges. For instance, liver and brain images with closed contours may have lost edges in certain directions. In contrast, the Canny algorithm suppresses unwanted edges and enhances sharpness, making it a more effective option [2].

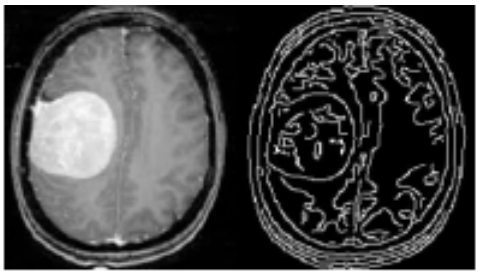

Fig. 1. Edge Detection

The limitation of the conventional system design on power consumption and performance has shifted its eyes to embedded processors. In the project, we put forward a hardware architecture for canny algorithm and processing system by utilizing the Zynq 7000 series architecture [3]. The architectural design provides hardware capability on SOC while enhancing the programmability of software. An efficient data reordering module for mapping the signals to the correct network is also and it attains low latency and high throughput. The entire system design utilizes the Zynq Processing system, DMA IP and custom IP generated for performing edge detection operation.

With both HLS and RTL, it is critical to understand the underlying architecture [4]. The RTL method offers flexibility and a low-level approach that can give the developer a design with improved performance. The developer's level of expertise in hardware design, however, has a significant impact on how efficient the design is. In this project, we will attempt to compare the differences in performance obtained from the implementation of Canny algorithm in HLS and HDL and analyze the causes for these differences.

III. Literature Survey

Canny edge detection algorithm shows reliable performance when used on noisy images, hence it is widely used for edge

detection applications. The advantage of canny edge detection is its ability to detect edges in the input image objects with a high localization rate, high accuracy, and low error rate [1]. But the high number of computations involved in this algorithm, especially when it is necessary to meet real-time constraints, results in higher power consumption and clock frequencies on general microprocessor architectures.

The state-of-the-art approach used in the FPGA implementation of Canny algorithm for edge detection [5] aims at utilizing the High-Level Synthesis (HLS) tool flow to accelerate its implementation on the Zynq platform by offloading the Canny algorithm from processing system (PS) to programmable logic (PL). Through hardware acceleration, the optimized implementation gives up to a 100x performance speedup. Another approach [1] uses HLS, with operations such as Gaussian smoothing, Sobel filtering, non-maximum suppression, double thresholding, and hysteresis developed on the Terasic DE-10 standard FPGAs development board.

Some studies have been conducted [4], [5] to analyze these discrepancies between HLS and RTL designs at the micro-architecture level by the implementation of the Sobel filter, Gaussian filter, and the Histogram design(used for thresholding applications). It is concluded that when the appropriate optimizations are used, HLS tools can generate efficient hardware. Even though HLS tools are evolving and becoming better at mapping the code to the hardware, code restructuring is still required. On the other hand, if the memory behavior is not carefully taken into account in the RTL design, it is highly probable that it will produce a functionally incorrect implementation. A faster implementation than the RTL version can be achieved in Vivado HLS by algorithmically eliminating the data dependency.

In the proposed project, we aim to use the learnings from these studies which analyze some stages used in the Canny algorithm(Sobel Filter, Gaussian Filter) and expand upon it to examine the causes for differences in the complete HLS implementation and HDL implementation of the Canny algorithm.

IV. Proposed Approach

The core of this architecture consists of 3 blocks and will be validated in both HLS and HDL approaches. Block 1 converts the RBG (BMP format) to a GRAY scale image based on the arithmetic mean of 3-pixel streams. The converted grayscale image is accessed by the programmable element by AXI-DMA IP. The second block is CANNY FILTER which detects the edges in the image. After Canny algorithm is performed, the PE returns the pixel stream to the DMA for storage in the microSD memory.

A. Preprocessing:

We need a grayscale image for loading on the processing core for performing double convolution operation in Block 2. We will discuss Block 2 in detail in further steps. For performing the grayscale conversion, we take the arithmetic mean of red, green and blue pixels.

Gray Pixel = (Red Pixel + Blue Pixel + Green Pixel) /3

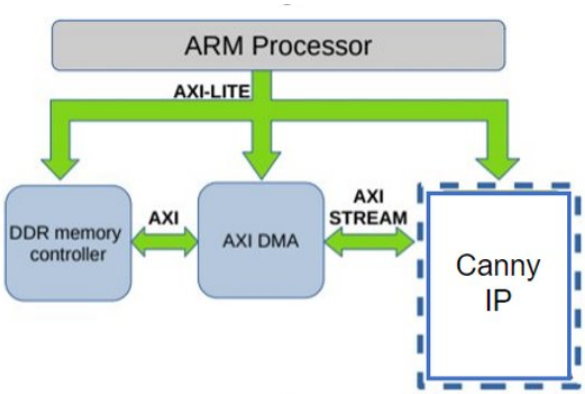

Fig. 2. Block Diagram

The preprocessing step is verified by Python/MATLAB-based testbench for RGB2GRAY conversion.

B. Line Buffer:

HDL implementation uses Line buffer and sliding window buffer structure [4], [5]. Storing and shifting previous pixel values can result in a significant improvement in speed by avoiding repeated accesses to local memory during kernel operations. This technique allows the kernel to access the required pixel values without accessing local memory more than once per pixel (Fig. 3.).

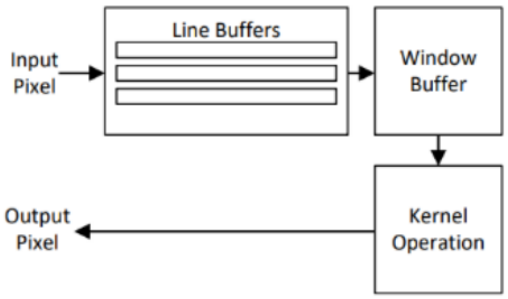

Fig. 3. Line buffer

Data access time is a significant bottleneck in FPGA applications, particularly when dealing with large images. To minimize data transfer between the FPGA and external memory, line buffers or row buffers are utilized to temporarily store the necessary rows for window operations.

To perform the same operation in Vivado HLS, array partitioning *pragmas* are used to specify how arrays should be mapped to the FPGA. This *pragma* offers several options, including type, factor, and dimension. To partition every element of a multidimensional array, the dimension option must be set to zero. Additionally, the unroll directive is required to create a parallelism between loop iterations.

C. Canny algorithm

The Canny edge detector algorithm as shown in Fig. 4. is implemented in four stages namely – Gaussian Blur Smooth-

ing, Gradient Computation, Non-Maximum Suppression and Hysteresis Thresholding [7], [8].

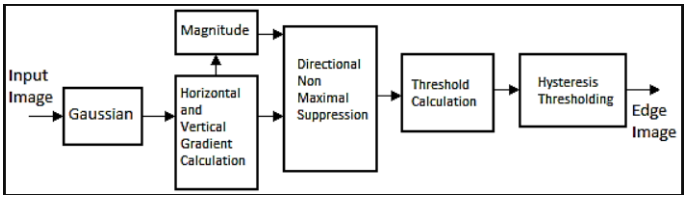

Fig. 4. Canny Algorithm

Gaussian Blur Smoothing – A Gaussian Filter is used to remove noise from the input image and smoothen it. This is required to remove unwanted edges from appearing in the output image. The 5x5 Gaussian mask is used for this process. We subtract the original image from the output image obtained. This is called Unsharp Masking.

Gradient Computation – To compute the edge strength matrix G and the edge direction edge angle matrix, we convolve the input matrix with the Sobel mask to obtain the first derivative in the horizontal and vertical direction. The values obtained are used to calculate the edge gradient and edge direction.

Edge gradient is given by :

$$G = \sqrt{Gx^2 + Gy^2} \tag{1}$$

Edge angle is given by :

$$\theta = atan2(Gy, Gx) \tag{2}$$

Non-maximum Suppression – The local maxima are found by comparing each pixel with its neighboring pixels along the direction of the gradient. The local maximum obtained is marked as an edge and the other values are suppressed. The objective of this process is to remove unwanted edges and create sharp edges. This process is also called edge thinning.

Hysteresis Thresholding – This is the final step in the Canny algorithm. Two threshold values are used to determine the intensity of the edge pixels. Pixels with values higher than the high threshold value are marked as strong, and those with values lower than the low value are suppressed. The pixels with values in between these threshold values are marked as weak.

The canny algorithm can be optimized in HLS by using *pragmas* for pipelining the dataflow and exploiting parallelism to achieve speedup and reduce hardware resource utilization. The HDL implementation needs to be carefully designed to avoid any memory dependencies and obtain an efficient, synthesizable RTL design.

D. Direct Memory Access

DMA can directly access the memory without a processor [4]. DMA controller acts as a bridge between memory and peripherals. At a time either DMA or processor can act as a master to access the system bus. The original picture is retrieved from the Processing System -DDR and sent to PL through DMA. Bus arbitration tells us which master can access the system bus memory. It is done based on round-robin arbitration where the most accessed master is given less priority.

E. Data Transfer

The original image on the external Double Data Rate memory is taken to the processing system PS block. Through the high-performance port, the segmented image stored in the memory heap goes to AXI direct memory access through smart connect. The smart connect is the latest modified version of AXI interconnect IP which eases the manual intervention in configuring master and slave. From smart connect, IP has to arbitrate between the processor and DMA.

For large data transfer between peripheral and memory, the processor configures the information in the internal DMA register and sets the DMA controller's control register to start the data transfer. Once the data transfer takes place, the status register is updated.

V. Methodology

*A. **HLS Implementation:***

Firstly, a C++ implementation of the Canny edge detection has been developed and checked for functional correctness. The testbench takes a 512*512 grayscale image obtained from the pre-processing unit as input and populates the input array. The top-level function reads the input array, performs edge detection processing and generates output image array.

The top-level function first applies gaussian blurring using a 7x7 Gaussian Filter. This involves convolution operations which have been optimized using loop tiling, array partitioning, and pipelining pragmas. The 2D input matrix is converted to a 1D matrix to provide faster memory access for convolutions.

Then gradient computation is done using sqrt and atan2 functions from math.h class for gradient magnitude and angle calculation in the c implementation. For the HLS implementation to be synthesizable, angle calculation is performed using python LUTs.

Non-maximum suppression function is applied to the resultant blurred matrix using the obtained gradient magnitude and direction matrices. Finally, hysteresis thresholding is performed taking 30% of maximum value as the threshold. The output matrix obtained from the top-level function is converted to an image showing detected edges by the testbench function.

The Canny Algorithm IP is exported and used to generate a bitstream and a hardware hand-off file for on-board testing using the Xilinx Vivado tool(fig. 5).

*B. **HDL Implementation:***

From the grayscale image obtained from Python preprocessing unit, we implemented the line buffer in Verilog to get the three lines of image pixels 3 * 512 and mask the lines with a window buffer that strides across the lines by 1. This above

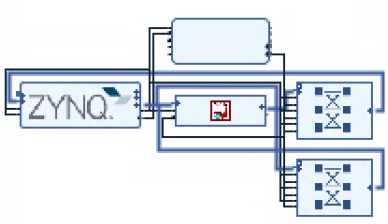

Fig. 5. Block Design

step of the line buffer is used because of limited BRAM in the PynqZ2 board.

Control logic: FSM to control the dataflow and the top-level module is called to instantiate the convolution, blurring, and line buffer modules. We also introduce the valid ready signal in the IP for DMA to interact for sending and accepting a new line of data. We map them to corresponding signals in PS-PL configuration and included the associated bus interface for packaging the IP.

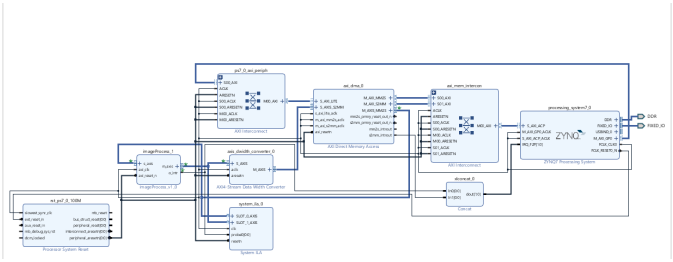

Fig. 6. Block Design

*1) **Block Design Flow**:* The image is stored in external DDR and the DMA controller is configured to stream four lines to the image processing IP for convolution. The same line is reused multiple times with a multiplexer. When a line is processed, an interrupt-based command is sent from the IP to the Zynq Processing system, triggering a service routine. The processed line data is then streamed back to the DMA controller, which interrupts the processor. The AXI stream width convertor is used to tackle the width mismatch between the DMA master port and Canny IP. The ACP port in PS is used to ensure cache coherence between memory and IP. This continues until the entire image is processed. The DMA controller sends the stream data from the IP to external DDR memory via the AXI4 interface. Debug pins are added to monitor signal flow using the ILA system debugger.

VI. Results

A. *HLS:*

Optimized latency for the Canny Algorithm IP is - 260ms.
Resource utilization for HLS:
BRAM – 81%
DSP – 6%
FF – 6%
LUT – 29%

B. *HDL:*

Line buffer is designed and tested by forcing the inputs in the timing waveform Fig.6.

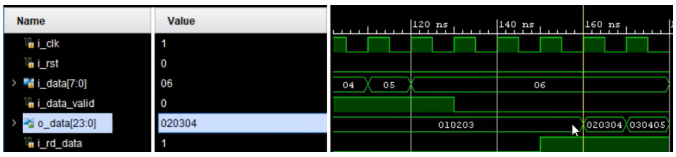

Fig. 7. Line Buffer Testing

Designed image processing IP block is tested with Lena image and Gaussian blurring along with Sobel filter to detect edges are applied. The block is synthesized and the timing diagram for input data and output data is shown in Fig.7. 4% LUT, 18% IO, and 1% Flip Flop are being utilized in the Synthesis report

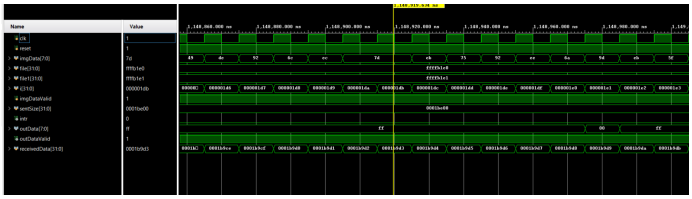

Fig. 8. Canny IP testing with Lena image

Once the canny block is tested, it is packaged into IP that can be instantiated in the block design. The following are resource and power utilization of system design in the Synthesis Report. The on-chip power of 1.3 Watts and LUT being 12%, FlipFlop 6% utilized in the system design with integration of Canny IP, AXI-DMA, and Zynq processing System

The obtained latency if the complete computation is carried out by the processing system is 6.451 sec. The obtained latency when run on PL fabric is 464.24 ms. Speedup is 14x faster.

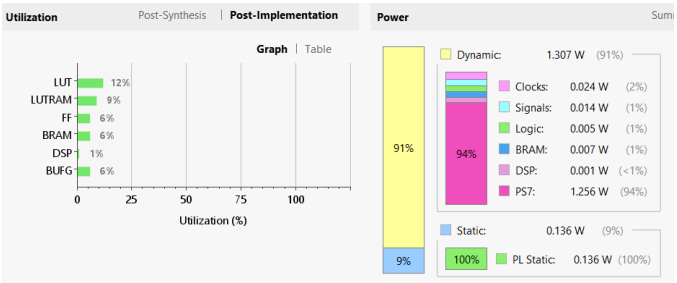

Fig. 9. Power and Resource Utilization

VII. MILESTONE COMPLETED - HLS & HDL

1) Preprocessing RGB to GREY
2) Gaussian Blur
3) Sobel kernel Filter
4) Functionally correct C code

5) Synthesizable implementation of Gaussian Blur, Non-Maximum Suppression and Hysteresis Thresholding using HLS
6) Synthesizable Functionally correct Verilog code

VIII. MILESTONE ACHIEVED - HLS & HDL

1) Hysteresis threshold - HDL
2) Synthesizable Gradient Computation - HLS
3) Optimization with # pragmas - HLS
4) On-board Implementation of HDL and HLS.
5) Comparative Analysis.

Fig. 10. Blurred image

Fig. 11. Canny Detected Image

IX. EXECUTION PLAN

Details	Start	End	Days
Estimated timeline			
Literature Survey	3/7/2023	3/12/2023	5
Functionally correct Synthesiable HLS code	3/13/2023	3/25/2023	12
HDL Code + Synthesis + Generate Bit stream	3/25/2023	4/10/2023	16
Optimization using Pragma and other tech	4/10/2023	4/24/2023	14
Integerate , Compare & Final Testing on Board	4/24/2023	5/2/2023	8

Fig. 12. Timeline

A. *Split of Work:*

- Literature Survey - Both
- Functionally correct Synthesizable HLS code - Saniya
- HDL Code + Synthesis + Generate Bitstream - Rithan
- Optimization using *pragma* and other tech - Both
- Integrate, Compare & Final Testing on Board - Both

X. CONCLUSION

In conclusion, this project has explored the implementation of the Canny algorithm for image processing using HLS and HDL methodologies to perform edge-detection with high accuracy. The two implementations have been analyzed to compare the latency and hardware utilization in both. Optimizations have been made using DMA memory access for HDL, and loop tiling, array partitioning etc. for HLS to improve latency and resource utilization. The results show that the HLS implementation provided a significant speedup of 1.78x compared to the HDL implementation. This highlights the potential benefits of using high-level synthesis tools for optimizing the performance of image processing algorithms. Further research can be conducted to investigate the use of HLS for other image processing algorithms and applications. Overall, this study emphasizes the importance of selecting the appropriate implementation methodology for a given application, and highlights the potential advantages of using HLS in the field of image processing.

XI. FUTURE WORKS

The developed Canny architecture can be used for medical applications such as MRI scans, fingerprint scans or other applications such as satellite imaging that require the use of edge-detection. This project can be extended from the current implementation for image processing to develop efficient architecture for video processing using Canny algorithm in the future. This project provides a good basis to extend and include other edge-detection algorithms for analysis using HLS and HDL implementations. Several algorithms such as Laplace, Prewett, Roberts etc. can also be tested to compare accuracy and efficiency and develop high-efficiency architecture for suitable applications.

REFERENCES

[1] Z. Tan and J. S. Smith, "Real-time canny edge detection on fpgas using high-level synthesis," 2020 7th International Conference on Information Science and Control Engineering (ICISCE), 2020.

[2] B. Li, J. Chen, X. Zhang, X. Xu, Y. Wei, and D. Kong, "A design of Zynq-based medical image edge detection accelerator," 2021 6th International Conference on Biomedical Signal and Image Processing, 2021.

[3] R. Dobai and L. Sekanina, "Image filter evolution on the Xilinx Zynq platform," 2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013), 2013.

[4] R. Millón, E. Frati, and E. Rucci, "A comparative study between HLS and HDL on SOC for image processing applications," Elektron, vol. 4, no. 2, pp. 100–106, 2020.

[5] H. M. Abdelgawad, M. Safar, and A. M. Wahba, "High Level Synthesis of Canny Edge Detection Algorithm on Zynq Platform," World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering, vol. 9, no. 1, 2015.

[6] M. Gurel, "A comparative study between RTL and HLS for image processing applications with fpgas," thesis.

[7] D. Daru, "Implementation of canny edge detection using hls(high level synthesis)," thesis.

[8] K. Yoshikawa, N. Iwanaga, T. Hamachi, and A. Yamawaki, "Development of fixed-point canny edge filter operations for high-level synthesis," The Proceedings of the 2nd International Conference on Industrial Application Engineering 2015, 2015.