Turkish Online Journal of Qualitative Inquiry (TOJQI)
Volume 12, Issue 7, July, 2021:1070 – 1080

Analysis of Edge detection using Zynq based SoC FPGA

Chintan S. Patel [a], Neol Solanki [b],Neel Tailor [c]

[a] Assistant professor, Birla Vishwakarma Mahavidyalaya, cspatel@bvmengineering.ac.in
[b,c] Student,Birla Vishwakarma Mahavidyalaya,neolsolanki@gmail.com, neelt430@gmail.com

Abstract

Image processing has applications in real-time in various fields, including feature recognition, forensics, military applications, and clinical so on. Image processing algorithms require a colossal amount of data to be stored since these algorithms operate on a plethora of array of pixels. Furthermore, the actual application requires these algorithms to be delivered at higher throughput and low latency, for instance, live video telecast and live object tracking. The parallelism and reconfigurable nature of FPGA and the high number of assets accessible on FPGA fill in as an ideal platform to accomplish this task in real-time. Edge detection is one of the crucial algorithms of image processing, after that, other algorithms such as object recognition, feature extraction depend. A great deal of researchers has aimed at various edge detection strategies. Several edge detection algorithms such as Roberts, Prewitt, Sobel, Scharr, and Canny are analyzed using the Xilinx Zed board and SoC design flow in this paper. The xfopenCV library has been utilized for edge detection, developed explicitly for Xilinx FPGAs and SoCs, can run up to 40 times faster than GPUs and 100 times faster than CPUs..

Keywords: FPGA, Image Processing, Edge Detection, Sobel, Canny, Robert, Prewitt, Laplacian of Gaussian, Marr-Hildreth, xfOpenCV, SoC Design flow, Zynq-7000, Zed Board

1. Introduction

The image processing algorithms operate on the data stored in a pixel frame, making it require a different approach than conventional microcontroller practice. A microcontroller sequentially executes the code, causing higher latency. To implement the major operation of image processing, convolution, on the microcontroller using assembly language, entails the loading of data from memory, moving data to registers, and then performing the convolution and finally storing the result back to the memory. This process would return as much as the number of pixels is present in the given image and increase power consumption and area due to a few names of peripherals handled on the microcontroller. Field programmable gate array (FPGA) is an ideal nominee for pragmatically achieving image processing as it works in parallel, steering a decline in latency. The cores accessible through FPGA, Digital Signal Processing (DSP) slices, Block RAM (BRAM), a Lookup table (LUT), industrial I/O standards, on-chip processor, and on-chip peripherals, can accomplish any digital design on a single chip. Figure 1 illustrates the difference between microcontroller and FPGA for several arithmetic operations. The parallel implementation needs only two clock cycles; on the other hand, sequential execution requires 12 clock cycles to complete [1]. Additionally, advanced zynq based FPGA provides the entrenched integration of software (processing system) and hardware (programmable logic).

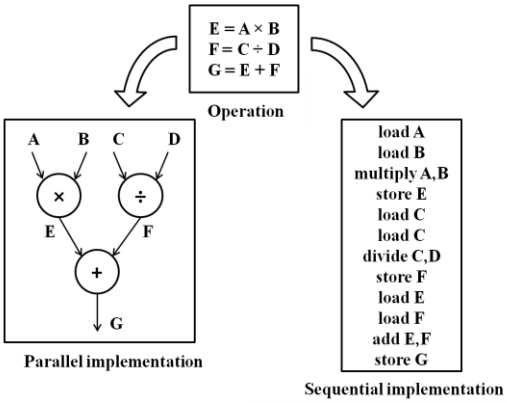


Figure 1. Parallel vs Sequential Implementation

However, designing with FPGA requires expertise in architectural details of the system at the Register Transfer Level (RTL) layer and profound knowledge of Hardware Description Language (HDL). Nevertheless, advanced Xilinx tools such as Intellectual Property (IP) integrator, High-level synthesis (HLS), System Generator provide automation at an individual level of abstraction. Many researchers implemented their diverse edge detection using these tools. Ex, R. K. Megalingam, M. Karath, P. Prajitha and G. Pocklassery performed Sobel edge detection algorithm with Vivado HLS and concluded that execution time for RTL is much less compared to the C specifications [4]. D. Sangeetha and P. Deepa proposed pipelining structure of canny edge detection using approximation method and performed output on Xilinx Virtex-5 FPGA [3]. B. C. Maheshwari, J. Burns, M. Blott and G. Gambardella implemented real-time Canny edge detection algorithm using vivado and demonstrated output on Pynq board [14]. Šušteršič T, Milovanović V, Ranković V, Filipović N, Peulić A designed Sobel and Robert edge detection using Xilinx system generator and compare them in terms of timing and resources utilization [15]. This paper aims to analyses SoC design flow in different aspects for first edge detection (Roberts, Prewitt, Sobel, and Scharr) to some advanced edge detection (Marr-Hildreth, Canny) methods.

2. Basic Edge detection

Identifying variations in the pixel value requires the computation of the first and second-order derivatives of arrays of pixels. The agent used to succeed this process is gradient, indicated by ∇, and defined as a vector of a given image, f, at coordinate (x, y),

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial x} \end{bmatrix} (1)$$

The vector has a central geometrical feature that indicates the largest variation of f at the position (x, y) per unit. M expresses the magnitude vector of ∇f (x, y),

$$\mathrm{M}(\mathrm{x},\mathrm{y}) = \sqrt{G_x^2 + G_y^2} (2)$$

M is the equivalent of the rate of change in the course of the gradient vector. G_x and$G_y$are known as gradient images. An angle,

$$\alpha(x,y) = \tan^{-1}\left(G_y/G_x\right) (3)$$

gives the direction of the vector measured on the x-axis.

Gradient Operatives

Partial derivatives $\partial f/\partial x \text{and} \, \partial f/\partial y$ need to be estimated at all pixel locations to capture the gradient image. Since we are dealing with digital quantities, a partial derivatives digital approximation is necessary over a neighborhood about a point. The equation for the digital derivatives is,

$$G_x = f(x+1,y) - f(x,y) (4)$$

$$G_y = f(x,y+1) - f(x,y) \; (5)$$

When we are dealing with diagonal edge detection, we need a 2D filter. The Roberts-cross edge gradient is one of the pioneer techniques to utilize a diagonal filter for edge detection. Consider the 3×3 region as shown in

figure 2 for reference,

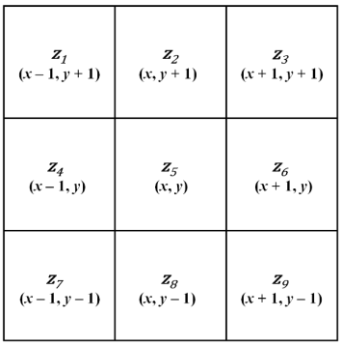


Figure 2. 3×3 region

Then, Roberts-cross edge filter is given by,

$$G_x = (z_9 - z_5) \quad (6)$$

$$G_y = (z_8 - z_6) \quad (7)$$

Kernels practiced to realize (6), (7) are,

$$G_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Although 2×2 is computationally simple, they are not symmetric about the center point $z_5(x, y)$, as witnessed in the above equations. The minimum size of the filter required to consider the information residing on the other side of the center point $z_5(x, y)$ is 3, can be represented as,

$$G_x = (z_7 - z_1) + (z_8 - z_2) + (z_9 - z_3) \quad (8)$$

$$G_y = (z_3 - z_1) + (z_6 - z_4) + (z_9 - z_7) \quad (9)$$

Kernel required to actualize (8), (9) are,

$$G_x = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

These two kernels are known as Prewitt operators. Sobel operator is a slightly revised version, with the added weight of 2 to the center coefficient.

$$G_x = (z_7 - z_1) + 2(z_8 - z_2) + (z_9 - z_3) \quad (10)$$

$$G_y = (z_3 - z_1) + 2(z_6 - z_4) + (z_9 - z_7) \quad (11)$$

Kernels adopted to implement (10), (11) are,

$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Scharr operator even considers the added weight of 3 and 10 to the side and centre coefficient, respectively.

$$G_x = 3(z_7 - z_1) + 10(z_8 - z_2) + 3(z_9 - z_3) \quad (12)$$

$$G_y = 3(z_3 - z_1) + 10(z_6 - z_4) + 3(z_9 - z_7) \quad (13)$$

Scharr kernels applied to fulfil (12), (13) are

$$G_x = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}, \quad G_y = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

Sobel and Scharr's operators provide image smoothing by putting weighs to the coefficients. The final output image is calculated by (2).

3. Advanced Edge detection

3.1 The Marr-Hildreth Edge detection

Marr-Hildreth operator incorporates two essential concepts. Firstly, intensity variations are dependent on image scale so that their edge detection needs the use of operators of various sizes. Secondly, an abrupt intensity value change would increase the highest point in the first derivation or similarly a zero crossing in the second derivation. Marr and Hildreth recommended that operator fulfilling these circumstances is $\nabla^2 G$, where ∇^2 is the Laplacian operator $\partial^2/\partial x^2 + \partial^2/\partial y^2$, the kernel for implementing laplacian operator is,

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

As the Laplacian and Gaussian are linear operators, it is feasible to apply the Gaussian filter and then calculate Laplacian of gaussian [6,7].

3.2 Zero-Crossing

One strategy for calculating zero-crossing at any pixel, of the filtered image is based upon using a 3×3 neighbourhoodcentred at the pixel. A zero-crossing at pixel shows that a minimum of two of its neighbourhood signs must be different. If two opposing pixels are getting compared to a threshold, then not only signs must differ, but also the absolute of their numerical must be bigger than the threshold magnitude [6].

3.3 Canny Edge Operator

Although this operator is very intricate, this operator's results are highly superior to those discussed first. Canny edge detection involves three essential concepts. First, low error rate-All the edges should be detected, and there should not be incorrect responses. Second, the edge should be localised appropriately; that is, the separation between the edge and the real edge should be minimal. Finally, the single edge point response- the local maxima around an edge point should be the lowest [3]. The canny operator takes the gradient and phase response of the image filtered with underlying edge detection kernels. Smoothing with a gaussian filter is necessary before performing the necessary edge detection. The gradient includes wide edges around the local maxima. Hence, the final step is to lean those edges. Non-maxima suppression performs this task for us [6].

3.4 Nonmaxima Suppression

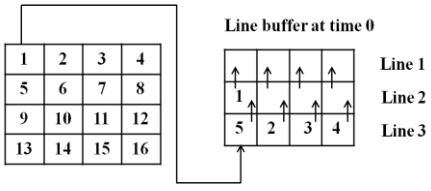
The principle of this approach is to define the discrete orientation of the gradient vector. Non-maximum suppression used to remove the incorrect response of edge detection—non-maximum suppression achieved by interpolating pixels for maximum accuracy [3].

3.5 Hysteresis

The final procedure is to threshold the non-maxima suppressed image to decrease incorrect edge points. Most of the image processing technique uses one threshold level. In that case, if the value of the threshold is meagre, there is still going to be some wrong edge (called as false positive); if the setting of the threshold is high, then original edge points are going to be removed (called as false negative). The Canny algorithm uses hysteresis thresholding, which uses two threshold values: a low threshold T_L, and high threshold T_H, to improve this situation [3,6].

4. Line Buffer and Window Buffer

A classic technique to effectuate the edge detection in hardware is to employ line buffer and window buffer. A line buffer carries memory elements capable of storing more than one line of the input image [5]. The kernel height specifies the number of rows in a line buffer. Since most of the edge detection algorithms, analyzed here, practice 3x3 kernels, the kernel height has assigned to 3. In HLS, xf::LineBuffer class can instantiate line buffer. The difference between xf and HLS class for line buffer is, xf has supplementary template arguments for describing the storage architecture to be inferred, and array reshape factor [8]. A window buffer is applied to save the current window's values and has an equal dimension as the kernel has.



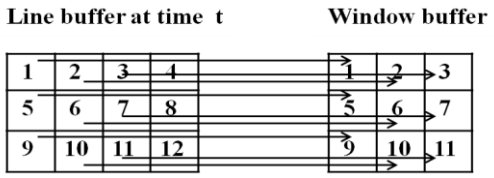


Figure 3. Line Buffer and Window Buffer

Figure 3 demonstrates the working method for line buffer and window buffer, assuming the image dimension as 4x4 for basic comprehension. The input image is scanned pixel by pixel into the line buffer in a streaming manner. For instance, pixel value 1 is copied to the first location; pixel value 2 is copied to the second location, and so on. Line buffer gets vertically shift while copying the input data and the most top-line buffer discharge data first. After period t, line 1, line 2 and line 3 gets filled.

5. Vivado HLS

The Vivado HLS tool integrates a C and C++ functions within an IP core without specifying RTL descriptions. Vivado HLS has tight integration with all of the design tools provided by Xilinx and IP cores and provides complete language assistance and optimized implementation for C/C++ algorithms [5]. Vivado HLS Tool requires a function written in C/C++ as a primary input; the principal function can carry a hierarchical chain of sub-functions. The constraint file specifies the FPGA target, clock period, clock uncertainty. Vivado HLS provides the facility to generate the different hardware implementation of the same C/C++ algorithms by providing optional directives without altering the functionality of code. Vivado HLS utilises C/C++ test bench before the synthesis phase to verify algorithm working and Co/RTL simulation to validate the generated design behavior before exporting it to as an IP core [9]. Figure 4 illustrates the workflow of the Vivado HLS.

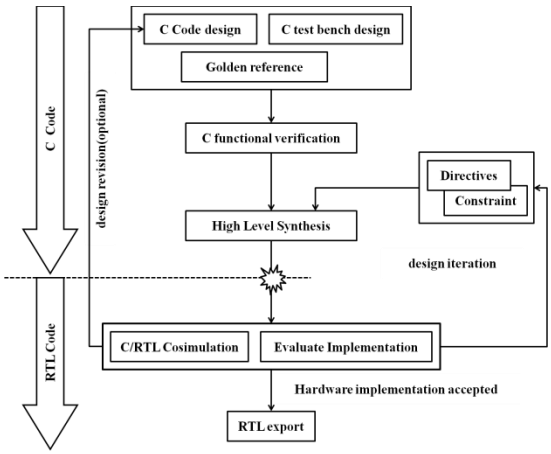


Figure 4. HLS Workflow

6. HLS AXI Protocol Simulation

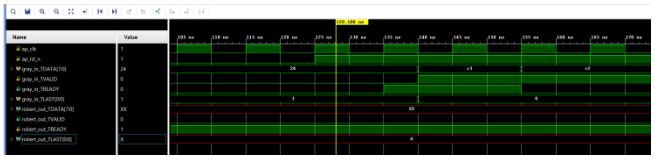


Figure 5. Robert Simulation(Input Signal Initialisation)

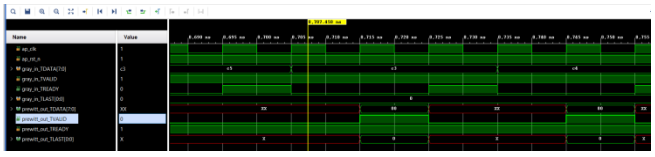


Figure 6. Prewitt Simulation(Output Signal Initialisation)

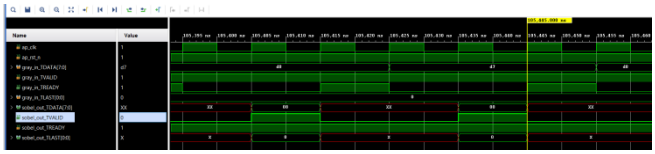


Figure 7. Sobel Simulation(Signals while running)

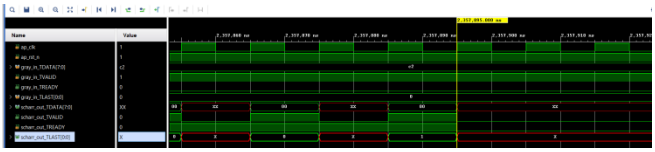


Figure 8. Scharr Simulation (Data transfer and signals)

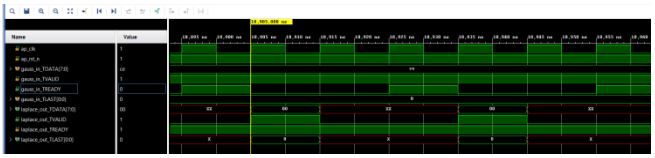


Figure 9. Laplace Simulation (Signals while running)

Figure 10. Zero Cross Simulation (Threshold specifications)

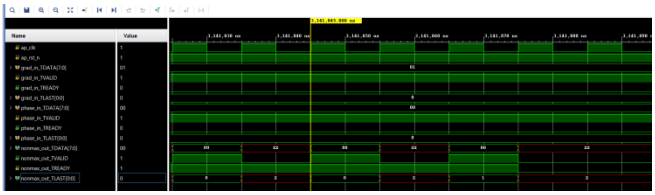


Figure 11. Non-Maxima Simulation (Signals while running)

Figure 12. Hysteresis simulation (Threshold specifications)

6.1 Advanced Interface Extensible (AXI) Protocol Handshaking

The AXI4-stream protocol consists of the signal TREADY, TVALID, TLAST, TID, TDEST, TKEEP, TSTRB, TDATA. For data transfer to initialize and terminate TREADY, TVALID, TLAST plays a vital role [13]. It is not plausible to display the entire start of frame (SOF) and end of the line (EOL) in one simulation picture. Therefore, we have distributed the data transfers events into the different simulation of the different filters for better comprehension. Edge detection IP begins operating when ap_rst_n signal is asserted by the zynq processing system, as shown in figure 5. After a few periods, gray_in_TVALID and gray_in_TREADY gets maintained, indicating that they are ready to receive the streaming of data through AXI DMA [12]. The signal gray_in_TLAST goes into the law state after a few clock cycles, reflecting the frame's start. This signal remains in this state until the data transfer ends. As illustrated in figure 6, the output signal prewitt_out_TDATA starts accepting the data when prewitt_out_TREADY and prewitt_out_TVALID are high. After both the input and output channel's TLAST signal goes to zero, the IP works, as shown in figure 7. The output is only registered when sobel_out_TREADY, sobel_out_TVALID are in high state and sobel_out_TLAST is in the low state. When the IP completes the operation, it asserts the output TLAST signal, scharr_out_TLAST in our case,

as shown in figure 8. For advanced edge detection methods, the required parameters such as threshold level are transmitted via the AXI-lite protocol. AXI-lite protocol contains the same signals as the AXI stream but has some additional signals such as ADDR, LEN, SIZE, BURST, LOCK, CACHE, PROT, QOS and REGION. The Zynq processing system writes the threshold value at the address indicated by the signal AWADDR once the WVALID (Write Valid) signal goes to high, as from figure 10 and 12.

7. Implementation

7.1 Hardware Implementation

The Zynq processing system operates as a master and slave through the general-purpose master and high-performance slave port. The image is present in the SD card in a bit map format. The zynq processing system receives the pixels from the image stored in the SD card. After reading the pixels from the image, the DDR (dual data rate) memory stores the pixels buffer. The AXI Direct Memory Access (DMA) IP works as a bridge for transmitting the pixels to the edge detection module in streaming mode. The AXI DMA maps the pixel's streaming into the memory at the address specified by the zynq processing system. Once the edge detection module completes the operation, it re-sends the pixels to the zynq processing system via AXI DMA after that AXI DMA maps the output pixels to the DDR memory [12]. For advanced edge detection methods like Marr-Hildreth and Canny, the Zynq processing system shares the threshold information via an AXI-lite protocol. The hardware system uses an AXI timer for measuring the time accurately, in 64-bit mode. Figure 13 portrays the high-level block representation of the hardware system.

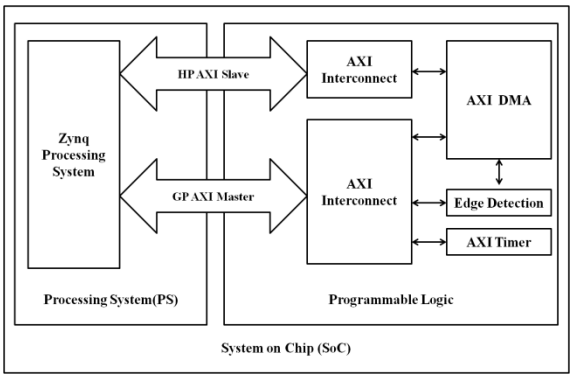


Figure 13. Hardware Implementation

7.2 Software Implementation

Subsequent designing the hardware, the next phase in the SoC design flow is to create a software code that runs on the hardware designed earlier. The drivers API initialize the drivers for using it in software. For detecting the image stored in the SD card, the FATFS module or Xilinx xilffs library is necessary. FATFS is a generic entity file system designed for the small embedded system. It is platform-independent. Image dimensions are 280×280 for removing the need to handle the padding present in the bit map format. For the advanced edge detection methods, Marr-Hildreth, the input is first blurred by the Gaussian filter with σ=1. The threshold level is 4 for zero-crossing detection. We have first blurred image with Gaussian filter σ=1 for canny edge detection. After applying gaussian filer, Sobel edge detection we applied, and then phase and magnitude response are calculated in the two different images. The Canny edge detection takes these two images as input. The low and high threshold levels are 20 and 60 for hysteresis, respectively. After the edge detection is over, the output image is written back to the SD card through FATFS.

8. Result and Discussion

To compare the output of basic and advanced level edge detection, we have used a Taj image of 280×280 size. From the output image, it can be seen that in basic level edge detection (figure 15,16,17,18), we get a better result in Scharr filter (figure 17) because it has higher value coefficients. On the other side, in advanced edge detection, we get better output in the canny filter because of its complex algorithm. It detects the smallest edges of the image. Table 1 shows that advanced edge detection requires more timing than basic edge detection, and we get faster output on FPGA than the MATLAB. Robert edge detection (figure 14) requires the smallest amount of resources corresponding to the other edge detection algorithms. The Canny (figure 20) and Marr-Hildreth (figure 19) require more resources because of its complexity. These basic level edge detections are used in barcode scanner and monitoring where accuracy is not a significant issue, but in medical image processing and defense applications, we required more advanced level edge detection like canny and Laplacian of gaussian

[15].

Figure 14. Input Image

Figure 15. Sobel Filter

Figure 16. Robert Filter

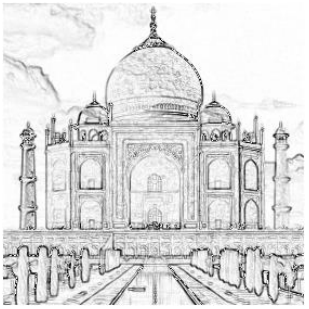


Figure 17. Scharr Filter

Figure 18. Prewitt Filter

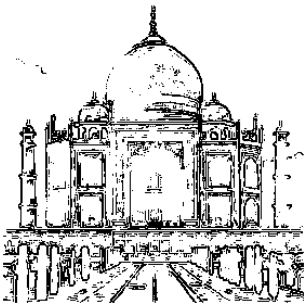


Figure 19. Zero-Crossing Filter

Figure 20. Hysteresis Filter

Table 1. Timing Information

Algorithm	Timing (Zed board)(100 MHz)(sec)	Timing (Matlab)(2.30 GHz)(i5)(sec)	Speed up factor
Robert	0.002692	0.0022632	×19
Prewitt	0.002692	0.0023811	×20
Sobel	0.002692	0.0024261	×21
Scharr	0.002692	0.0024572	×21
Marr-Hildreth	0.006168	0.0179565	×8
Canny	0.006354	0.020562	×8

Table 2. Resource Utilization

Algorithm	LUTs	Registers	Block RAM Tile
Robert	3595	4373	3
Prewitt	3714	4459	3.5

Sobel	3717	4460	3.5
Scharr	3870	4520	3.5
Laplacian	3627	4427	3.5
Zero Cross	3681	4426	3.5
Non-Maxima	4401	5370	5.5
Hysteresis	3644	4437	3.5

9. Conclusion and Future Work

This paper correlated the Xilinx zynq SoC FPGA design flow with the sequential flow for the sophisticated edge detection: image processing algorithms. Using the parallelism and hard IPs present on the FPGAs, latency is decreased compared to the microprocessor/microcontroller approach. We utilized higher abstraction level tools like HLS and xfopenCV library for implementing the algorithms on the hardware. Additionally, different directives accessible via HLS reduced the design latency and improved the optimization level. The line and window buffer provide great flexibility for implementing most image processing algorithms on the FPGAs or SoC boards. The AXI protocol lessened the communication overhead between different IPs in the design. By practicing the zynq based SoC design flow, we managed some assets of the hardware (PL) design through the zynq processing system (PS) with the help of the AXI protocol. The real video processing system can use these IPs in their design with the help of the other supplementary AXI IPs, such as AXI subset converter, AXI Video Direct Memory Access (VDMA), AXI Video Graphics Array (VGA) core, and video timing controller by running at the required pixel resolution clock. For future work, we are looking forward to using custom IPs in real-time applications with the camera interface with the help of zynq SoC FPGA..

References

[1] L. H. Crockett, R. A. Elliot, M. A. Enderwitz and R. W. Stewart, The Zynq Book: Embedded Processing with the ARM CortexA9 on the Xilinx Zynq-7000 All Programmable SoC, First Edition, Strathclyde Academic Media, 2014.

[2] Xing J., Tan W., Bai J. (2020) Design of Object Edge Detection System Based on FPGA. In: Jia Y., Du J., Zhang W. (eds) Proceedings of 2019 Chinese Intelligent Systems Conference. CISC 2019.

[3] D. Sangeetha and P. Deepa, "An Efficient Hardware Implementation of Canny Edge Detection Algorithm," 2016 29th International Conference on VLSI Design and 2016 15th International Conference Embedded Systems (VLSID), Kolkata, 2016, pp. 457-462.

[4] R. K. Megalingam, M. Karath, P. Prajitha and G. Pocklassery, "Computational Analysis between Software and Hardware Implementation of Sobel Edge Detection Algorithm," 2019 International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 2019, pp. 0529-0533.

[5] A. Cortes, I. Velez and A. Irizar, "High level synthesis using Vivado HLS for Zynq SoC: Image processing case studies," 2016 Conference on Design of Circuits and Integrated Systems (DCIS), Granada, 2016.

[6] R.C. Gonzalez, R.E.Woods: Digital Image Processing, 3rd edition, Prentice-Hall, pp.748-976, 2007

[7] Sunmin Song, SangJun Lee, Jae Pil Ko and Jae Wook Jeon, “A Hardware Architecture Design for Real-time Gaussian filter,” 2014 IEEE International Conference on Industrial Technology (ICIT), Feb. 26 - Mar. 1, 2014, Busan, Korea

[8] Xilinx Inc., Xilinx OpenCV User Guide. UG1233(v2019.1), June 5,2019

[9] Xilinx Inc., Vivado Design Suite User Guide, High-Level synthesis. UG902(v2018.3), December 20, 2018.

[10] N. Gaikwad and V. N. Patil, "Verification of AMBA AXI On-Chip Communication Protocol," 2018, Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India

[11] S. K. Mohapatra, B. R. Swain and S. K. Mahapatra, "Optimised approach of sobel edge detection technique using Xilinx system generator," 2015 2nd International Conference on Electronics and Communication Systems (ICECS), Coimbatore, 2015.

[12] Xilinx Inc., AXI DMA v7.1- LogiCORE IP Product Guide, Vivado Design Suite.(PG021), June 14,2019

[13] Xilinx Inc., AXI Timer v2.0- LogiCORE IP Product Guide, Vivado Design Suite. (PG079), October 5, 2016

[14] B. C. Maheshwari, J. Burns, M. Blott and G. Gambardella, "Implementation of a scalable real time canny edge detector on programmable SOC," 2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA).

[15] Šušteršič T., Milovanović V., Ranković V., Filipović N., Peulić A. (2020) Medical Image Processing Using Xilinx System Generator. In: Filipovic N. (eds) Computational Bioengineering and Bioinformatics. ICCB 2019