Design of Medical Image Hardware Acceleration Platform by SDSoC for ZYNQ SoC

Liang Mu, Tao Wei, Yuyu Tao, and Chang Liang
School of Computer, Electronics and Information, Guangxi University, Nanning, China
Email: alvismuliang@163.com, 429197597@qq.com, yuyu.tao@foxmail.com, 1659382158@qq.com

Xuejun Zhang
Guangxi Key Laboratory of Multimedia Communications and Network Technology, Nanning, China
Email: xjzhang@gxu.edu.cn

***Abstract*—In this paper, a design scheme for hardware acceleration for ZYNQ SoC programming and medical image processing using SDSoC development software is introduced and compared with traditional hardware language programming and Vivado HLS programming. In SDSoC, developers can use C/C++ language for hardware development as well as OpenCV and xfOpenCV library. OpenCV is a widely used image processing library, which can not only to shorten the development cycle and reduce the difficulty of hardware development, but also take up more FPGA resources. XfOpenCV is an OpenCV library optimized by Xilinx, similar in usage to OpenCV. In this paper, several typical images processing algorithms are used to test medical image data onto DICOM format, and the design of the proposed scheme and the traditional scheme is compared with the perspectives of processing speed comparison, power consumption, and development cycle. Sobel filter, Gaussian smoothing, and Harris corner detection are chosen for comparison study for their widely usages in image processing. Finally, performance on four platforms - CPU, ARM, ZYNQ and GPU are compared and evaluated to our method.**

***Index Terms*—ZYNQ, SDSoC, medical image processing, hardware acceleration**

I. Introduction

With the continuous improvement in medical image processing technology, the effect of medical image data processing is getting better and better. In recent years, more 3d medical image processing methods has been emerging [1].

However, the more optimized algorithm is accompanied by the continuous increase in the amount of code. If the Hardware Description Language (HDL) is used, the amount of code will increase geometrically. The main reason are that the hardware description language is aimed at specific hardware, similar to There are few libraries such as OpenCV, which leads to a long development cycle using hardware description languages, and the use of hardware description hardware for medical image processing is not widely used.

At the same time, the real-time processing performance has been low due to the large amount of processed medical images. Moreover, both image data and processing algorithms may require a certain degree of confidentiality. The use of hardware for medical image processing acceleration can solve the above two problems well. Hardware acceleration can process data onto parallel, and the processing speed is greatly accelerated. The code of the hardware device does not run on the PC side. As a black box, it is not easy to be cracked and maintains good confidentiality. It can be seen that using hardware to accelerate medical image processing is a better optimization solution.

Given the above problems, this paper proposes a medical image processing platform based on ZYNQ, which can accelerate the medical image processing hardware. ZYNQ SoC is a SoC that integrates ARM and FPGA from Xilinx. It has two ARM9 cores. This design uses the SDSoC development environment to develop the ZYNQ SoC. The advantage of the SDSoC tool is that it can use C/C++ language for hardware development without writing a hardware language at all. SDSoC can also use the OpenCV library [2], which is widely used by image processing program developers, greatly reducing the difficulty of development and shortening the development cycle. Although using the OpenCV library takes up more resources than not using it, from the perspective of comprehensive consideration of the development cycle cost and the use of hardware resources, using the library development method can significantly reduce development costs.

The key to hardware development of SDSoC is that it integrates with Xilinx's high-level synthesis tool Vivado HLS. The HLS tool is a code synthesis technology. The HLS described in this article refers specifically to the HLS applied to Xilinx FPGAs. Traditional FPGA development tools use register transfer layer code synthesis, which is low-level and can only use hardware description languages. And HLS are a high-level description, you can use C++ and other higher-level languages.

Prototypes of high-level synthesis tools have appeared in the 1980s and early 1990s, such as Hebe [3], HAL [4], MIMOLA [5], ADAM [6], Hyper [7] popularized. It

Manuscript received July 4, 2020; revised November 3, 2020.

doi: 10.18178/joig.8.4.98-106

really started to flourish in the 1990s, when mainstream EDA vendors improved RTL tools (behavioral Compiler from Synopsys, Monet from Mentor Graphics [8] and Visual Architect from Cadence [9]) [10], the first commercial HLS tool was born. Since then, HLS tools have been widely popularized and applied to many fields, such as aerospace applications [11], real-time video processing [12], etc.

The main function of the HLS tool is to reduce the development time of the hardware description language. We can see the obvious effect of the two pictures (Fig. 1 and Fig. 2) in the official Xilinx documentation.

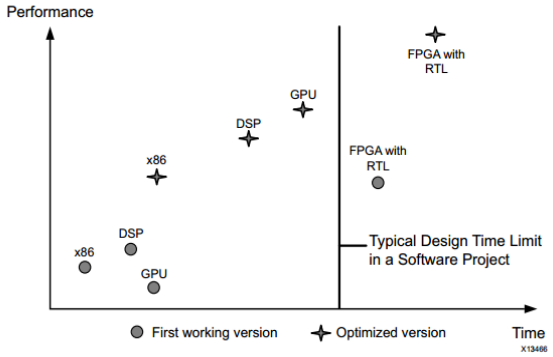


Figure 1. Design time vs. application performance with RTL design entry.

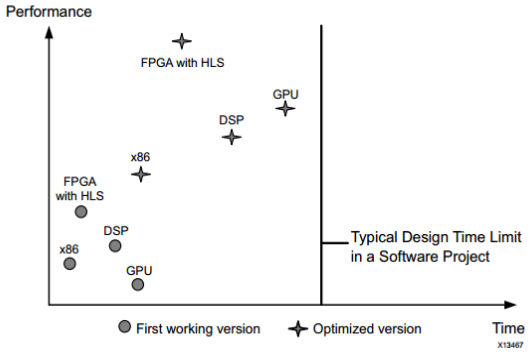


Figure 2. Design time vs. application performance with Vivado HLS compiler.

From the comparison of Fig. 1 and Fig. 2, it can be seen that the development time of FPGA using traditional RTL tools is much longer than that of DSP and GPU, while the development time of FPGA using HLS tools is even close to that of x86, and has the highest application performance.

This article introduces several common and somewhat complex image processing algorithms, and uses DICOM format medical images as data sources, using SDSoC development tools, and implemented on the ZYNQ XC7z015 platform. The first part is the comparison of Sobel algorithm for medical images of different platforms and different acceleration schemes. The second part are the comparison of different schemes for Guassian Filter algorithm, and the comparison between this design and the scheme for the paper [10], discussing the advantages and disadvantages. The third part are the comparison of different schemes for Harris corner detection algorithm. The design in this paper is compared with the scheme for [13], and the advantages and disadvantages are discussed. The rest of this article is structured as follows. In the second section, the specific resources of the ZYNQ SoC and the characteristics of the SDSoC tool are described. In the fourth section, the full text is summarized, and some conclusions are drawn.

II. ZYNQ AND SDSOC

The hardware platform designed in this article is Xilinx's ZYNQ XC7z015 SoC, which integrates ARM and FPGA at the same time: two ARM9 (PS part) and FPGA logic unit (PL part) are integrated, and the execution sequence of the main program is controlled by ARM It runs the operating system, and the FPGA part performs the parallel operation of the algorithm. Between ARM and FPGA, the AXI4 bus protocol is used for high-speed data transmission. In this design, the Ubuntu 16.4 operating system is running on one of the ARM9.

The development tool adopts the SDSoC development environment. The SDSoC environment provides a framework of developing and delivering hardware-accelerated embedded processor applications using standard programming languages [14]. The SDSoC environment provides a framework of developing and delivering hardware-accelerated embedded processor applications using standard programming languages. SDSoC is a development environment provided by Xilinx for the SoC platform. SDSoC integrates three development tools: SDx, Vivado, and Vivado HLS. When the platform template is available, only the SDx tool can complete the entire development process. Using C/C++ programming in SDx (the design in this article uses C++ programming), You can use the OpenCV library. If you want to accelerate the program by hardware, you only need to select the corresponding function of the software. If you want to improve the hardware acceleration efficiency, you need to use Vivado HLS related syntax to add acceleration code to the function. Unlike writing general C/C++ code, writing HLS code requires keeping the hardware in mind and always paying attention to hardware limitations. There are specific pragmas instructions in HLS to further to specify the code to achieve higher control over the synthesis process. If the constraints are not met, you need to modify the programs instruction. The following is an example of a programs instruction that implements a pipeline:

```
#pragma HLS LOOP_TRIPCOUNT min=1 max=COLS/NPC
#pragma HLS LOOP_FLATTEN off
#pragma HLS PIPELINE
```

It is worth mentioning that Xilinx provides a hardware-specific xfOpenCV library. This library uses a large number of pragmas instructions, and the official fully utilizes the acceleration performance of pragmas instructions. Of course, the xfOpenCV library is not as comprehensive as the OpenCV library, but the two libraries can be cross-compiled.

Unlike the traditional Vivado HLS direct development of IP cores, after compiling in SDx, SDSoC will automatically generate Vivado HLS IP cores and Vivado projects and create Block Design, copy the generated files to the SD card of the ZYNQ hardware device. Run the program, which further reduces development time.

III. Case Studies

In this section, three typical medical images processing algorithms will be introduced: Sobel Filter, Gaussian Filter, Harris Corner Detection. As a case study of medical image processing hardware acceleration, the main reason for using them are that they are the most frequently used algorithms among many medical images processing algorithms such as medical image segmentation and medical image registration. As a comparison, each algorithm is implemented using four methods to discuss the advantages and disadvantages of the solution to this article. The algorithms in the program all use the OpenCV library.

The first step of the medical image processing program is to read medical images. Since the OpenCV library is used for image processing, the DICOM images read are converted into Mat images (Mat is the data structure used to store images in OpenCV). After analyzing the DICOM image format information, we wrote a DICOM image of Mat image program and called it as a general program interface. The DCMTK library is not used in the program to convert DICOM images. This is to make the program have better compatibility and portability. On some SoCs, some problems will occur when the DCMTK library is compiled, so this article has written the conversion program. The following three algorithms are converted by this program before execution, and then the medical image data can be processed.

A. Sobel Filter Implementation

1) Introduction to sobel filter algorithm

Sobel filters are often used to extract horizontal edges (horizontal features) and vertical edges (vertical features) of grayscale images. The Sobel operator contains two sets of 3x3 matrices, which are horizontal and vertical, and convolve them with the image plane to obtain the approximate values of the horizontal and vertical approximate brightness difference. If A represents the original image, and Gx and Gy represent the images detected by the horizontal and vertical edges respectively, below is the formula:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \times A \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \times A$$

The approximate horizontal and vertical gradients of each pixel of the image can be combined with the following formula to calculate the size of the gradient:

$$G = \sqrt{G_x^2 + G_y^2}$$

2) The experimental scheme

In medical image processing, the Sobel filter is often used as the first step of the algorithm. As a pixel-level processing algorithm, the Sobel filter also consumes considerable computing time for medical image processing. Therefore, the Sobel filter is implemented on hardware It can accelerate image processing to a certain extent.

In order to conduct sufficient comparison and verification, the following four methods will be used for Sobel medical image processing. They all use the OpenCV library. The input image is the same. It is a medical image of DIOCM format with a size of 512×512 and a pixel width of 16 Bit, as shown in Fig. 3. They are: (1) Use C++ language programming on the PC side and use the CPU to run the program (hereinafter referred to as method 1); (2) Use C++ language programming on ZYNQ without using function hardware acceleration; (3) Use C++ language programming on ZYNQ, Use hardware acceleration for the Sobel function; (4) Use Python programming on the PC side and use GPU to run the program. The reason for using Python is to shorten the development cycle. If you use C++ for GPU programming, it will increase development time.

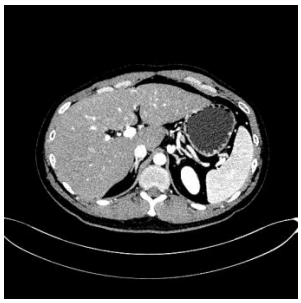
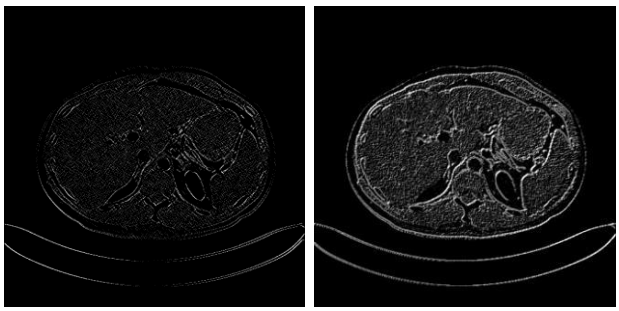
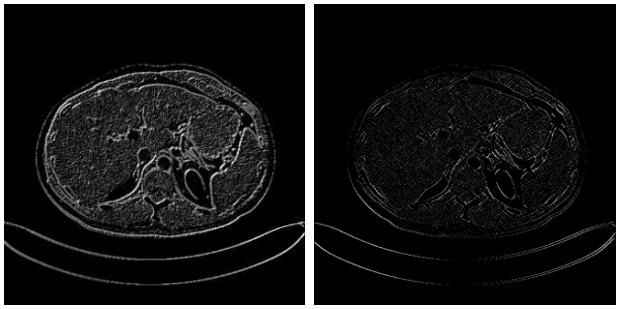


Figure 3. Input DICOM image



(a) Output image of method 1. (b) Output image of method 2.



(a) Output image of method 3. (b) Output image of method 4.

Figure 4. Output image after Sobel processing.

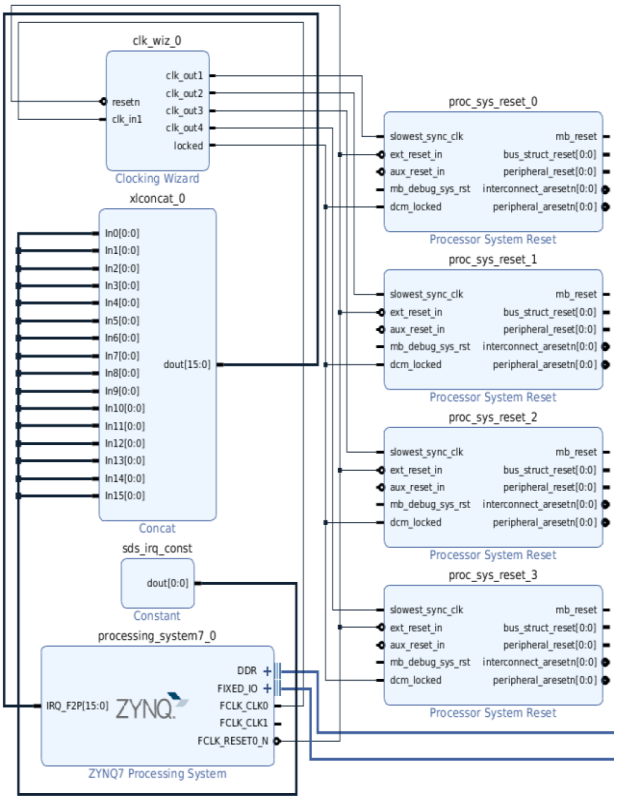
The development tool used in Method 1 is Microsoft Visual Studio 2019. The program first converts the DICOM images to a Mat image and then uses the Sobel functions as OpenCV to process the image and output the runtime. The output image is shown in Fig. 4(a), and the running time statistics are shown in Table I.

The development tool used in Method 2 is SDSoC. Method 2 runs a program similar to Method 1 on ZYNQ. The difference is that the library used in SDSoC is xfOpenCV. The output image of Method 2 is shown in Fig. 4(b). The Vivado tool integrated into the SDSoC development environment can automatically generate Block Design and can also generate detailed resource usage and power consumption reports. The Block Design of Method 2 is shown in Fig. 5(a), and the report result is shown in Fig. 6(a).

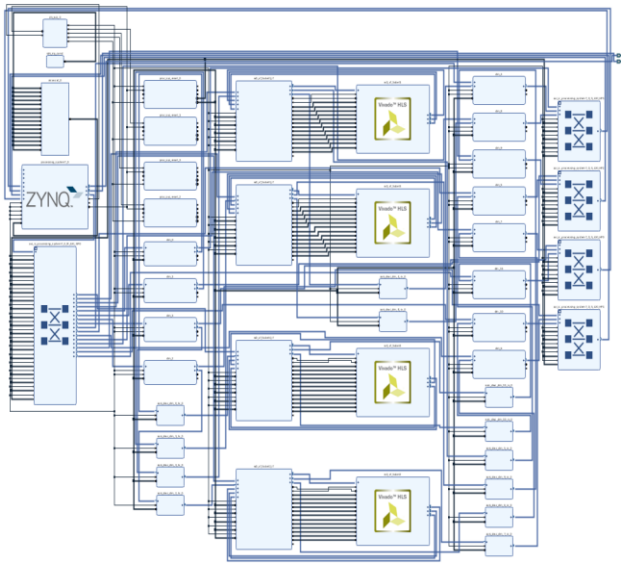
Method 3 is the design method introduced into this article. The development tools and procedures of Method 3 and Method 2 are the same. The xfOpenCV library is also used. The difference is that Method 3 selects the Sobel functions as the hardware acceleration function of SDSoC. The output image of Method 3 is shown in Fig. 4(c), Block Design is shown in Fig. 5(b), and the report results are shown in Fig. 6(b).

Method 4 runs on the PC. The development tool used is the Spyder tool for Anaconda Navigator. Similarly, the OpenCV library is also used and accelerated by the GPU. The specific models and parameters are listed in Table I. The operation result is shown in Fig. 4(d).

3) Experimental results and analysis

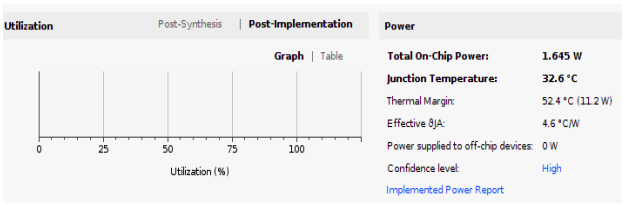


(a) Block design of method 2.

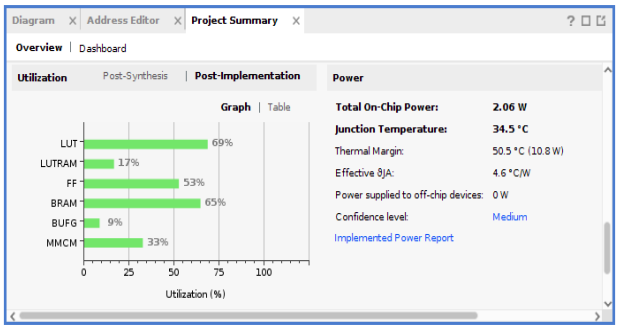


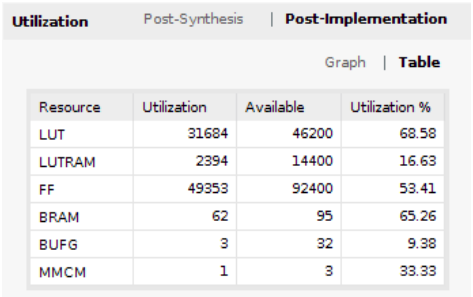
(b) Block design of method 3.

Figure 5. Block design of method 2 and method 3.



(a) Project summary of method 2.





Resource	Utilization	Available	Utilization %
LUT	31684	46200	68.58
LUTRAM	2394	14400	16.63
FF	49353	92400	53.41
BRAM	62	95	65.26
BUFG	3	32	9.38
MMCM	1	3	33.33

(b) Project summary of method 3.

Figure 6. Project summary of method 2 and method 3.

It can be concluded from the comparison of the four images of Fig. 4 that the different methods of use will not affect the results of the image processing. It can be seen from Fig. 5(a) that there is no IP core generated by the HLS tool for Block Design, which means that there is no hardware acceleration. At this time, the program is completely running in the ARM part of the ZYNQ SoC, that is, at this time ZYNQ SoC is equivalent to an ARM microcontroller. As can be seen from Fig. 5(b), there are 4 IP cores (parts with Vivado HLS icon) generated by HLS in Block Design, which means that hardware acceleration has been used.

In Table I, we compare the processor model, operating frequency, development cycle, operating environment, operating speed and power consumption of the four methods (SDSoC function hardware acceleration used in Method 3 is referred to as SDSoC-hard), from the table in comparison, we can compare the advantages and disadvantages of each method. Since Method 3 is the method mainly introduced into this article, the following will compare Method 1, Method 2, Method 4, and Method 3.

TABLE I. COMPARISON TABLE OF SOBEL FILTER IMPLEMENTATIONS IN DIFFERENT METHODS

	Method1 (x64)	Method2 (SDSoc)	Method3 (SDSoC-hard)	Method4 (GPU)
Platform	Intel i7-8700 CPU	ZYNQ XC7z015	ZYNQ XC7z015	GTX-1080 GPU
Clock Frequency (MHz)	3200	150	150	1607-1733
Development Time (man-days)	0.5	0.7	0.7	0.5
Processing Time(ms)	15	243.42	2.73	4.17
Power Consumption (W)	>65	1.645	1.864	>180

TABLE II. METHOD 3 COMPARED WITH THE REFERENCE [10] DATA

Graphics	**RTL Designer**	**SW Designer**	**Method3(SDSoC-hard)**
Platform	ZYNQ XC7z020	ZYNQ XC7z020	ZYNQ XC7z015
Clock Frequency (MHz)	200	200	150
Development Time (man-days)	12	1	0.7
Processing Time (ms)	0.398	0.498	2.73
Image Size	320×240	320×240	512×512
Number of Slice LUTs	1111	21946	31684
Number of Slice Registers	748	25439	49353
Number of Block RAMs	3	18	62

First, compare method 1 and method 3. Compared with CPU, ZYNQ runs at a speed difference in 0.015/0.0044=3.4 times, and the processing speed has a very significant improvement. In terms of power consumption, the power consumption of method 4 is only 2.01W, and the lowest power consumption of method 1 is 65W. The difference in power consumption is 65/2.01=34.87 times. The huge power consumption gap can often bring greater power to design products. Development space and economic benefits. Comparing the development cycles of Method 1 and Method 3, there is not much difference between the two. Overall, Method 3 is obviously more advantageous than Scheme 1.

Next, compare Method 2 and Method 3. Since method 2 runs on ARM, its speed is the slowest, and other parameters are the same, but method 2 cannot be executed in parallel, and the frequency is always much lower than method 1 and method 4, so this method is the same as the usual use The method of ARM is basically the same, only has advantages in power consumption, but the processing time is too long. In general, method 3 has advantages over method 2.

Finally, compare method 4 with method 3. Method 4 uses the GPU. GPU is the commonly used hardware acceleration method. Due to its parallel data processing structure, it is much faster than CPU in processing large amounts of data. Therefore, although method 4 consumes twice as much power as method 1, it still has More advantages than Option 1. But compared to method 3, the power consumption gap is too large, with a difference of 120/2.01=59.7 times. In large-scale data centers, this gap is very important.

In addition, when querying related designs, It was found that [10] also conducted a similar study. Here, the same algorithm case of [10] can be used as a comparison. The comparison objects of [10] are RTL Designer (Vivado HLS) and SW Designer (Vivado HLS). With OpenCV), and compare some of its data, as shown in Table II. The data onto Table II shows that SDSoC uses xfOpenCV and function hardware acceleration compared to Vivado HLS with OpenCV, which can further to shorten the development time and save resources such as Slice LUTs, Slice Registers, Block RAMs, etc. However, In contrast, Block RAMs are not There is not much reduction, because the program can make more reasonable use of SoC resources during sub-automatic synthesis, and more programs instructions are used in xfOpenCV to allocate Block RAMs resources, which can improve data throughput. The data onto the table also shows that [10]'s Vivado HLS with OpenCV has a shorter processing time. This is mainly because (1) [10] and method 3 have different image sizes, and the amount of data are different (320×240)/(512×512)=3.413 times; (2). The difference in clock frequency is 200/150=1.33 times. Of course, the resource occupancy of the two solutions is higher than that of RTL Designer, but under the huge development time advantage, the resource occupancy is very cost-effective. Based on the comparison of data onto Table II, It is obvious that the design of this design has more advantages.

B. Gaussian Filter Implementation

1) Introduction to Gaussian filter algorithm

Gaussian filtering is a linear smoothing filter, suitable for eliminating Gaussian noise, and is widely used in the

noise reduction process of image processing. It can be simply understood that Gaussian filter denoising is the weighted average of the pixel values of the entire image. The value of each pixel is obtained by the weighted average of its value and other pixel values in the neighborhood.

The specific operation is: use the template (or convolution, mask) specified by the user to scan each pixel in the image and use the weighted average gray value of the neighborhood pixels determined by the template to replace the center pixel of the template, as shown in Fig. 7.

The two-dimensional Gaussian distribution formula is as follows:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

0.0453542	0.0566406	0.0453542
0.0566406	0.0707355	0.0566406
0.0453542	0.0566406	0.0453542

(a) 3×3 Gaussian kernel coordinates. (b) Gaussian kernel weight matrix.

0.0947416	0.118318	0.0947416
0.118318	0.147761	0.118318
0.0947416	0.118318	0.0947416

(c) Gaussian kernel weight matrix with sum of 1.

Figure 7. Gaussian kernel weight calculation matrix.

2) The experimental scheme

The Gaussian filter also uses four methods of comparison. The four methods are exactly the same as the four methods of 3.1 above. So, this paper won't repeat them here, and are still referred to as methods 1-4. For a better comparison, the input image is still shown in Fig. 3. As the results of the four output images are basically the same, only the output results of SDSoC with xfOpenCV (Method 3 in 3.1 in the text) are shown here, shown in Fig. 8.

As mentioned in 3.1, Method 2 only runs on ARM of hardware connections, so Block Design and Project Summary are the same as in 3.1, as shown in Fig. 5(a) and Fig. 6(a). The Block Design and Project Summary of Method 3 is shown in Fig. 9 and Fig. 10. Method 3 uses a 3-core design, which is determined by the on-chip resources of ZYNQ XC7z015. The limitation lies mainly in the number of LUTs. As can be seen from Fig. 10, the utilization rate of LUTs has reached 70%.

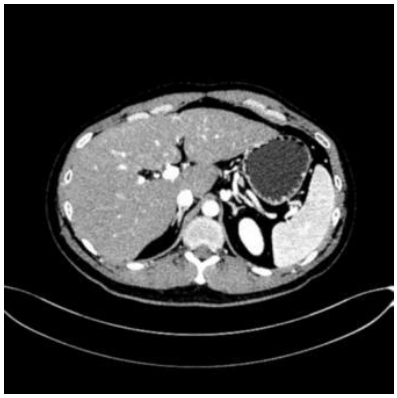


Figure 8. Output image of Gaussian filter.

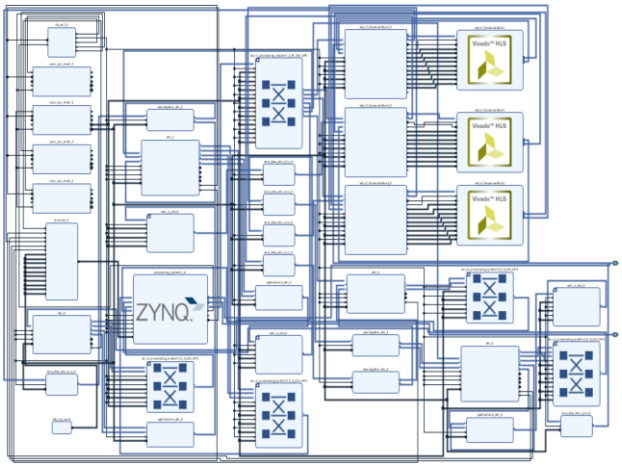
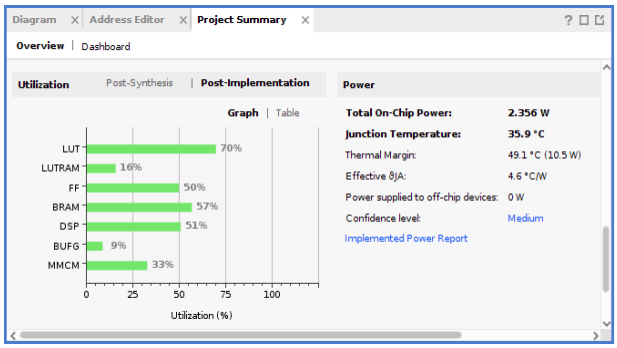


Figure 9. Block design of method 3.



Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	32271	46200	69.85
LUTRAM	2267	14400	15.74
FF	46296	92400	50.10
BRAM	54	95	56.84
DSP	81	160	50.63
BUFG	3	32	9.38
MMCM	1	3	33.33

Figure 10. Project summary of method 3.

3) Experimental results and analysis

Compared with method 1 and method 3, method 3 has a speedup of 18/3.646=4.9369 times and saves 65/2.356=27.589 times of power consumption. Method 3 still has great advantages compared to method 1.

The conclusion obtained by comparing method 2 with method 3 is the same as that in 3.1, and will not be repeated here.

Compared with Method 3, the processing time is the same, and the gap is narrowed. As mentioned above, this is because the resources of Method 3 are limited. In terms of power consumption, method 3 achieves 180/2.356=76.4 times power saving, which still has huge advantages.

Similarly, this article found a similar experiment in another paper [13], using its experimental data as a comparison, the data statistics are in Table III. In [13], Gaussian kernels of various sizes are used for comparison. The Gaussian kernel used in this paper is 5×5, so the 5×5 data in Table IV in [13] is used for comparison. It can be seen from the data onto the table that method 3 in this paper has a greater advantage in speed, which is mainly due to the difference in image size. The size ratio of the image data is converted to the time ratio. The speed of the two schemes is the same. [13] The development tool used is Vivado HLS. According to the comparison data onto 3.1, Method 3 in this paper has the advantage of a short development cycle.

TABLE III. COMPARISON TABLE OF GAUSSIAN FILTER IMPLEMENTATIONS IN DIFFERENT METHODS

	Method1 (x64)	Method2 (SDSoc)	Method3 (SDSoC-hard)	Method4 (GPU)
Platform	Intel i7-8700 CPU	ZYNQ XC7z015	ZYNQ XC7z015	GTX-1080 GPU
Clock Frequency (MHz)	3200	150	150	1607-1733
Development Time(man-days)	0.5	0.7	0.7	0.5
Processing Time(ms)	18	287.67	3.646	3.34
Power Consumption (W)	>65	1.645	2.356	>180

TABLE IV. METHOD 3 COMPARED WITH THE REFERENCE [14] DATA

Graphics	Vivado HLS	Method 3
Platform	ZYNQ XC7z020	ZYNQ XC7z015
Clock Frequency(MHz)	157	150
Kernel Size	5×5	5×5
Processing Time(ms)	13.209	3.646
Image Size	1920×1080（8-bit per pixel）	512×512（16-bit per pixel）
Number of Slice LUTs	20656	32271
FFs	17724	46296
Number of Block RAMs	2	54

C. Harris Corner Detection Implementation

1) Introduction to Harris corner detection algorithm

Harris Corner Detection is often used in medical image registration algorithms. The basic idea of Harris algorithm is to use a fixed window to slide into any direction on the image, and compare the degree of change of the pixel gray scales in the window before and after the sliding. If the sliding in any direction has a large gray scale change, then we can think that there are corner points in the window. The Harris image corners detection algorithm is summarized as follows, divided into the following five steps: (1). Calculate the gradient of the image of X and Y. (2) Calculate the product of the gradients in the two directions of the image. (3) Use Gaussian functions to perform Gaussian weighting (take sigma=1) to generate elements A, B, and C of matrix M. (4) Calculate the Harris response value R of each pixel, and set R that is less than a certain threshold t to zero. (5) Perform non-maximum suppression in a 3×3 or 5×5 neighborhood, and the local maximum point is the corner point in the image.

2) The experimental scheme

This section also uses a similar structure of the previous two sections, and also uses four methods to achieve Canny edge detection. Some conclusions are found through the comparison of different methods. The four methods are the same as the four methods of 3.1 above, which will not be repeated here, and are still simply referred to as methods 1-4. The results of the four output image are the same, and only one output image is displayed-Fig. 11.

3) Experimental results and analysis

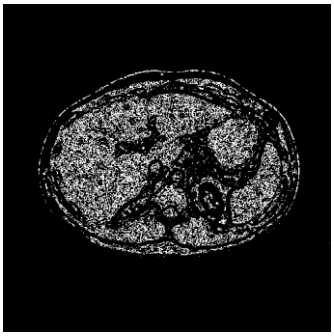


Figure 11. Output image of Canny edge detection.

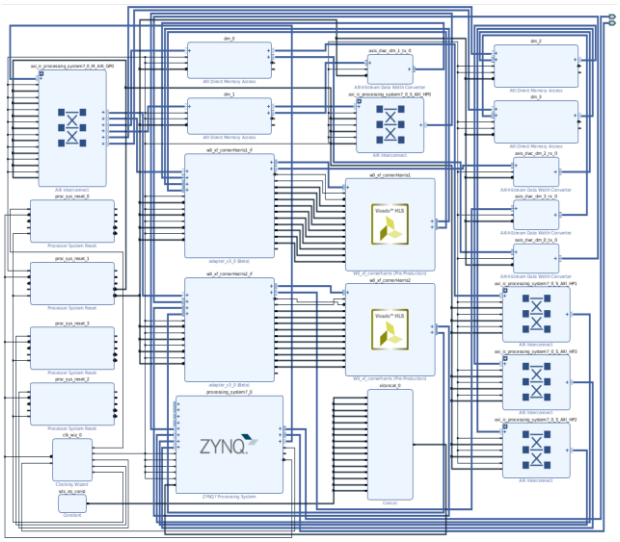
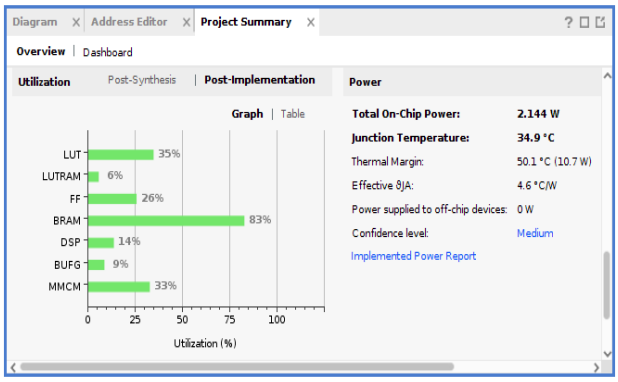


Figure 12. Block design of method 3.



Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	16224	46200	35.12
LUTRAM	850	14400	5.90
FF	23777	92400	25.73
BRAM	79	95	83.16
DSP	22	160	13.75
BUFG	3	32	9.38
MMCM	1	3	33.33

Figure 13. Project summary of method 3.

TABLE V. COMPARISON TABLE OF HARRIS CORNER DETECTION IMPLEMENTATIONS IN DIFFERENT METHODS

	Method1 (x64)	**Method2 (SDSoc)**	**Method3 (SDSoC-hard)**	**Method4 (GPU)**
Platform	Intel i7-8700 CPU	ZYNQ XC7z015	ZYNQ XC7z015	GTX-1080 GPU
Clock Frequency (MHz)	3200	120	120	1607-1733
Development Time(man-days)	0.5	0.7	0.7	0.5
Processing Time(ms)	18	335.43	6.61	3.8
Power Consumption (W)	>65	1.645	2.144	>180

As can be seen from Fig. 12, Method 3 uses a 2-core design, which is determined by the limitations of on-chip resources. It can be seen from Fig. 13 that the use of BRAM has reached 83%, which is the key to limiting the number of parallel algorithm cores.

The data comparison of the four methods is shown in Table V, and it can be seen that the processing speed of Method 3 is still much faster than that of the CPU, achieving an acceleration of 18/6.61=2.72 times. However, as the frequency of the ZYNQ SoC decreases, the processing time of Method 3 is pulled away by Method 4, but this does not mean that Method 3 is inferior to GPU in complex algorithm processing. ZYNQ XC7z015 is just a low-end chip in the Xilinx chip. Xilinx's high-end chips are fully capable of complex algorithm processing and can achieve faster processing speed through a multi-stage parallel connection.

IV. CONCLUSION

This paper uses SDSoC development environment to implement three common algorithms in medical image processing on ZYNQ SoC. The important significance of the research is that the hardware acceleration of medical image processing greatly shortens the processing time of medical images and improves the real-time nature of medical image processing. Medical image processing is an interdisciplinary subject, and the requirements for developers are high. Another important significance of this article is the reduction of the development threshold for medical image processing. SDSoC can use the OpenCV library and the xfOpenCV library. Among them, OpenCV is familiar to the majority of software designers, which greatly shortens the development cycle of the project. The usage of xfOpenCV and OpenCV is very similar. There is no development threshold for developers familiar with OpenCV. The xfOpenCV is officially "#pragma" instruction has been optimized for library functions, running faster and saving hardware resources. In this paper, the Sobel Filter, Gaussian Filter, Canny Edge Detection three algorithms are compared using four methods, after testing on four platforms CPU, ARM, ZYNQ, GPU, the data are compared and discussed. It can be seen that the use of SDSoC for hardware acceleration of medical image processing on ZYNQ SoC is the most advantageous, mainly reflected on/off: fast running speed, short development cycle, low power consumption, difficult situations in traditional hardware development. Next, the design scheme for this article is undoubtedly a good way to solve the problem. At the same time, the design scheme for this article has good scalability, which can use multiple ZYNQ SoCs in parallel, or use SoCs with more on-chip resources. The design in this article mainly uses dual cores. In the case of sufficient resources, more cores can be used for parallel computing, and the processing time is almost unchanged. This means that more cores can double the operating speed. Of course, the power consumption will also There is a certain improvement.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

During the process of writing this paper, Liang Mu was responsible for writing the paper and designing the hardware and software; Dr. Zhang directs the experiment; Wei Tao was responsible for the analysis of experimental data; Yuyu Tao helped modify the software program; Chan liang helped revise the paper. All authors had approved the final version.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (Grants Nos. 81460274 and 81760324); and Natural Science Foundation of Guangxi (Grant No. 2017JJA170765y).

REFERENCES

[1] S. E. Mahmoudi, *et al.*, "Web-based interactive 2D/3D medical image processing and visualization software," *Computer Methods and Programs in Biomedicine*, vol. 98, no. 2, pp. 172-182, 2010.

[2] G. Bradski, "The opencv library," *Dr Dobb's J. Software Tools*, vol. 25, pp. 120-125, 2000.

[3] A. P. Chandrakasan, *et al.*, "HYPER-LP: A system for power minimization using architectural transformations," in *Proc. IEEE/ACM International Conference on Computer-aided Design*, 1992.

[4] P. G. Paulin, J. P. Knight, and E. F. Girczyc, "HAL: A multi-paradigm approach to automatic data path synthesis," in *Proc. 23rd ACM/IEEE Design Automation Conference*, 1986.

[5] G. D. Micheli, *et al.*, "The Olympus synthesis system," *IEEE Design & Test of Computers*, vol. 7, no. 5, pp. 37-53, 1990.

[6] J. Granacki, D. Knapp, and A. Parker, "The ADAM advanced design automation system: Overview, planner and natural language interface," in *Proc. 22nd ACM/IEEE Design Automation Conference*, 1985.

[7] P. Marwedel, "The MIMOLA design system: Tools for the design of digital processors," in *Proc. 1st Design Automation Conference*, 1984.

[8] J. P. Elliott, *Understanding Behavioral Synthesis: A Practical Guide to High-Level Design*, Springer Science & Business Media, 1999.

[9] A. Hemani, *et al.*, "Application of high-level synthesis in an industrial project," in *Proc. International Conference on Vlsi Design*, 1994.

[10] A. Cortes, I. Velez, and A. Irizar, "High level synthesis using Vivado HLS for Zynq SoC: Image processing case studies," in *Proc. Conference on Design of Circuits and Integrated Systems*, 2016.

[11] P. J. Pingree, *et al.*, "Implementing legacy-C algorithms in FPGA co-processors for performance accelerated smart payloads," in *Proc. IEEE Aerospace Conference*, 2008.

[12] K. Denolf, S. Neuendorffer, and K. Vissers, "[IEEE 2009 International Conference on Field Programmable Logic and Applications (FPL) - Prague, Czech Republic (2009.08.31-2009.09.2)] 2009 International Conference on Field Programmable Logic and Applications - Using C-to-gates to program streaming image processing kernels efficiently on FPGAs," 2009, pp. 626-630.

[13] O. Shipitko and A. Grigoryev, "Gaussian filtering for FPGA based image processing with high-level synthesis tools," in *Proc. IV International Conference on Information Technology and Nanotechnology*, 2018.

[14] V. Kathail, *et al.*, "SDSoC: A higher-level programming environment for Zynq SoC and Ultrascale+ MPSoC," in *Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016.

Liang Mu was born in Taiyuan, Shanxi, China in 1993. He received the B.S. degree in Measurement and Control Technology and Instrumentation from the College of Instrumentation and Electronics, North University of China in 2017, Taiyuan, Shanxi. He is currently pursuing the M.E. degree with the school of Computer and Electronic Information, Guangxi University, majoring in electronic and communication engineering. The current research area is FPGA hardware acceleration of medical image processing.

Xuejun Zhang received the B.S. degree in Physics from Guangxi University, PR China in 1991; the M.S. degree in Electronics and Information Systems Engineering from Gifu University, Japan, in 2001; and the Ph.D. degree in Electronics and Information Systems Engineering from Gifu University, Japan, in 2004. From 2004 to 2007, he continued his research in Computer-Aided Diagnosis as a postdoctoral researcher at Department of Intelligent Image Information, Division of Regeneration and Advanced Medical Science, Graduate School of Medicine, Gifu University. Since 2007, he was a Professor of School of Computer and Electronic Information, Guangxi University, Nanning, Guangxi, China. His research interests include computer aided diagnosis system, image segmentation, pattern recognition, visualization in medicine. He has published over 70 papers in Journals, Proceedings, Book chapters and Scientific Magazines.

Tao Wei was born in Hechi, Guangxi, China in 1994. He received the B.S. degree from Wuhan University, Wuhan, Hubei, China, in 2017. He is currently pursuing the M. E. degree with the School of Computer, Electronics and Information, Guangxi University, Nanning. His research area is Deep Learning and Medical image processing.

Yuyu Tao was born in Guangxi, China in 1994. He received the B.E. degree in Electronic science and technology from the School of Physical Science and Technology, Guangxi University, Nanning, China in 2017, where he is currently pursuing the M.E. degree with the School of Computer, Electronic and Information, Guangxi University, Nanning, China. He current research areas include the design of the optical fiber surface plasmon resonance sensors and their applications.

Chan Liang was born in Yulin, Guangxi, China in 1995. She received the B.S. degree from Guangxi University, Nanning, Guangxi, China in 2018. She is currently pursuing the M.S. degree with the School of Computer, Electronics and Information, Guangxi University, Nanning. Her research area is Medical image processing.