Proceedings of International Conference on Technology and Social Science 2024(ICTSS 2024)

Optimization of Image Usage for Sprite Drawing Hardware Combining High-Level Synthesis and FPGA Internal Memory

Keigo Aoki[1, a] and Akira Yamawaki[1,b]

[1]Department of Electrical Engineering and Electronics, Kyushu Insutitute of Technology, 1-1 Sensui-cho, Tobata-ku, Kitakyushu City, Fukuoka, 804-8550, Japan

[a]<aoki.keigo980@mail.kyutech.jp>, [b]<yama@ecs.kyutech.ac.jp>

Keywords: sprite, fpga, internal memory, image replacement, dma

Abstract. We are developing a sprite drawing hardware as one of the functions in the game libraries suitable for high-level synthesis automatically converting software to hardware. Normally, images and other data used in games are stored in large external memory, but reading from external memory may badly affect the performance due to time-consuming. To solve this problem, we developed a sprite drawing hardware that uses the internal embedded memory of an FPGA for image storage and combines it with software optimized for high-level synthesis. However, due to the small capacity of the internal embedded memory, large number of images cannot be stored into. Therefore, this paper proposes a system that performs image replacement in the background of a sprite drawing to handle more images. The replacements strategies are software-based and hardware-based versions. This paper performs a comparative evaluation among two strategies. The experimental results show that the hardware-based version is more than 7 times faster than the software-based version.

1. Introduction

Mobile devices, such as smartphones and tablets, are becoming more and more popular, requiring higher performance and functionality as well as more power savings. Mobile devices run a variety of applications, and each application requires different processing. It is not realistic to implement hardware that can handle all of them in a mobile device. Therefore, we propose a mobile device that uses an FPGA (Field Programmable Gate Array) to achieve high performance and low power consumption using hardware to perform the heavy processing load [1].

The hardware design of FPGAs is usually done at the register transfer level (RTL) using a hardware description language (HDL). However, RTL design has difficulties not found in software design, such as the need to design processing in clock units and the need to consider the frequency of operation and timing of signal arrival. Therefore, describing complex processing in RTL is not only complicated and costly, but also may cause bugs. Thus, we develop hardware using high-level synthesis, which automatically converts software into hardware [2,3]. However, ordinary software programs assume software processing. Therefore, high-level synthesis of such programs may generate inefficient hardware. Thus, we have developed a software library for high-level synthesis that considers the hardware organization [4-8]. The algorithm of the process to be converted to hardware finds the continuity of memory access, parallelism of data processing and so on. By automatically creating a description suitable for high-level synthesis tools to find these factors, it is possible to achieve high processing performance with low power consumption, which is unique to hardware. As part of this research, we have also developed a software library for sprite drawing processing [9, 10].

In sprite drawing, sprite images and background images are combined and then displayed as a single screen. The images used in games, such as sprites and background images, are usually stored in a large external memory. Therefore, it is necessary to read the images through an external port during processing, which is expected to take a long time. Circuits for transferring images to and from external memory are also required. Thus, we developed a sprite drawing hardware that utilizes the internal memory of FPGA to store the images used in games [11]. As a result, the circuit size was reduced, and power efficiency was improved. However, the capacity of FPGA internal memory is very small compared to external memory, so it is considered insufficient for storing images used for games.

Therefore, in this paper, we developed an image replacement system to enable the use of more images in sprite drawing hardware that utilizes FPGA internal memory. We developed software-based and hardware-based image replacement strategies and compared the performance of each.

This paper is organized as follows. Section 2 explains the overall process flow of sprite drawing and its description for high-level synthesis, and hardware organization that utilizes the FPGA internal memory. Section 3 shows the proposed image replacement strategies. Section 4 performs the experiments and discusses the experimental results. Finally, Section 5 concludes the paper.

2. Sprite Drawing Hardware

2.1 Basic Algorithm of Sprite Drawing

In sprite drawing, first, the coordinates of the image to be drawn on the screen are specified. Second, the pixel of the sprite image is read, and it is determined if the pixel has color data. Then, if the pixel has color data, it is written to the screen. If the pixel is transparent, it is not written, and the background image data is written to the screen.

Fig. 1 shows an example of the sprite drawing process. The 0th pixel of the sprite image has no color data, so the background image data is written to the screen. Next, the 1st pixel has color data, so its data is written to the screen. In this way, a composite image of the sprite and background is displayed on the screen.

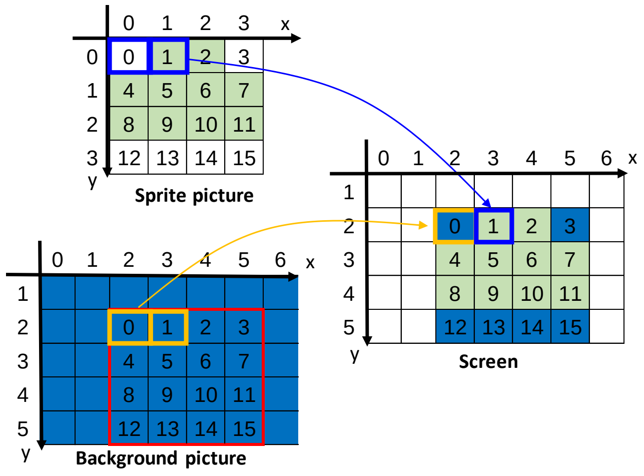


Fig. 1. Sprite drawing algorithm.

2.2 Software Program for High-Level Synthesis and Hardware Organization

The sprite drawing algorithm described in 2.1 does not take hardware organization into account. Therefore, a software program was developed to parallelize processing by taking advantage of the features of hardware processing [10]. Furthermore, we improved the software program along with the development of a hardware organization that uses internal memory for storing the sprite images [11]. Fig. 2 shows its software program for high-level synthesis, and Fig. 3 shows its hardware organization.

The following describes the process flow. First, the pixel data of the sprite image and background image are read, and the sprite image data is judged whether it exists or not, as described in 2.1. Then, the sprite image data or background image data is stored in the line buffer. This is the first stage of processing and Loop1 of the program shown in Fig. 2. Next, the data stored in the line buffer is written to the screen data. This is the second stage of processing and Loop2 of the program shown in Fig. 2. By repeating these stages of processing, the sprite image and background image are composed.

```
 1: void Loop1(int i, uint32_t *sp, uint16_t x, uint16_t y, uint16_t w,
 2:                 uint32_t *bg, uint16_t ofst, uint32_t buf[MAX_SPRITE_WIDTH]){
 3:     L11: for( int j = 0; j < w; j++) {
 4:             uint32_t pix[2];
 5:             pix[0] = sp[i * w + j + ofst];
 6:             pix[1] = bg[( i + y ) * GAME_SCREEN_WIDTH + x + j];
 7:             if( pix[0] != 0 ) buf[j] = pix[0];
 8:             else          buf[j] = pix[1];
 9:     }
10: }
11: void Loop2(int i, uint16_t x, uint16_t y, uint16_t w, uint32_t *fg,
12:                  uint32_t buf[MAX_SPRITE_WIDTH]){
13:     L12: for( int j = 0; j < w; j++ ) {
14:             fg[( i + y ) * GAME_SCREEN_WIDTH + x + j] = buf[j];
15:     }
16: }
17: void SpriteDrawHW( uint32_t *sp, uint16_t x, uint16_t y, uint16_t h,
18:                      uint16_t w, uint32_t *fg, uint32_t *bg, uint32_t ofst){
19: #pragma HLS INTERFACE mode=bram  port=sp storage_type=ram_1p
20: #pragma HLS INTERFACE mode=m_axi bundle=d0 port=bg offset=slave
21: #pragma HLS INTERFACE mode=m_axi bundle=d0 port=fg offset=slave
22:     uint16_t i, j;
23:     L1: for(int i = 0; i < h; i++) {
24: #pragma HLS DATAFLOW
25:             uint32_t buf[MAX_SPRITE_WIDTH];
26:             Loop1(i, sp, x, y, w, bg, ofst, buf);
27:             Loop2(i, x, y, w, fg, buf);
28:     }
29: }
```

Fig. 2. Software program for sprite drawing.

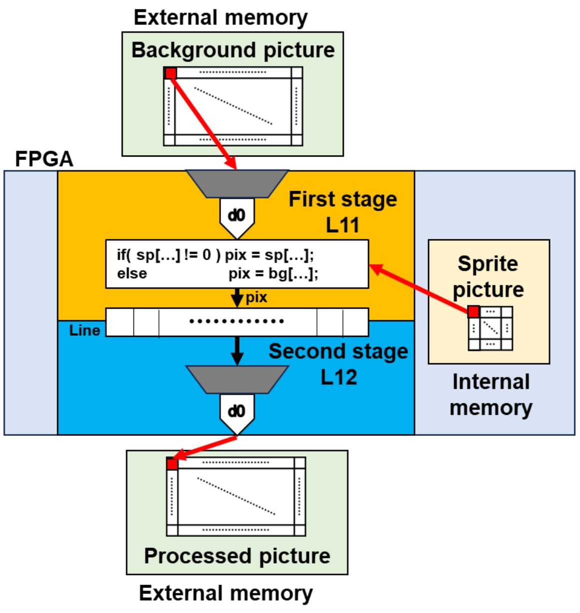


Fig. 3. Hardware Organization of Sprite drawing.

If nothing is specified, these two stages are normally processed in sequence. However, they are processed in parallel by using the DATAFLOW pragma as shown in line 24 of the program shown in Fig. 2. In addition, the hardware organization is such that the sprite image is stored in the internal memory and the background image is stored in the external memory.

3. Image Replacement Strategies

We have proposed a hardware organization for sprite drawing that uses internal memory for image storage. Hardware utilizing internal memory has been found to be more power efficient than using burst transfers for reading from external memory [11]. However, compared to external memory, internal memory in an FPGA has an extremely small capacity. For example, the maximum internal memory capacity of the FPGA used in this study is 560 KB. Therefore, in the case of a 128 x 64 size sprite image, only about 17 images can be stored at most. Thus, it is not realistic to store all the sprite images used in a game. Furthermore, the capacity of internal memory in an FPGA depends on its FPGA. If a larger capacity of internal memory is required, a more expensive and larger size of FPGA is needed. However, this is not desirable for the widespread use of mobile devices using an FPGA. Thus, in this study, we develop image replacement systems that can handle more images in the proposed hardware organization, even with small internal memory capacity. We developed two types of system: a software-based version and a hardware-based version.

Fig. 4 shows the hardware organization of the software-based version, and Fig. 5 shows that of the hardware-based version. In the software-based version, the sprite image data rewriting of the internal memory is controlled by the embedded CPU of the FPGA. In contrast, in the hardware-based version, we developed a DMAC (Direct Memory Access Controller). The external memory can be read directly from the hardware side by the DMAC, and the image data in the internal memory is rewritten.

In the internal memory, several sprite images can be stored. Therefore, the next required image can be replaced with an unused image in the internal memory at the same time as the sprite drawing process. Thus, in the two proposed versions, the image replacement is processed in parallel with the sprite drawing process, as shown in Fig. 6. In this way, the image replacement time may be able to hide in the processing time of sprite drawing. As a result, the number of images that can be handled is expected to increase even though the processing time is unchanged.

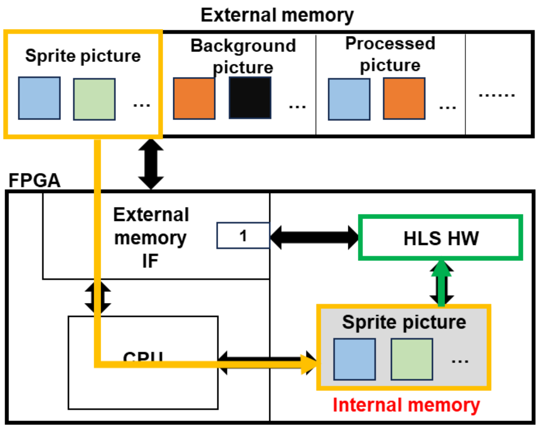


Fig. 4. Software-based version.

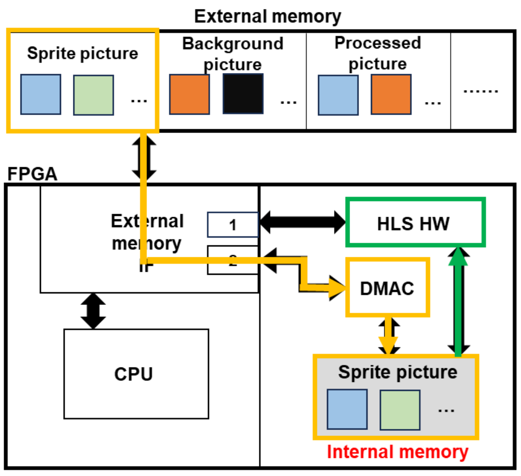


Fig. 5. Hardware-based version.

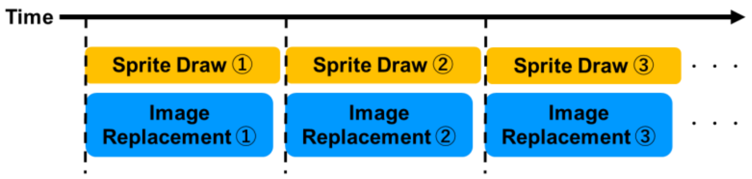


Fig. 6. Overview of sprite drawing and internal memory.

To achieve this, the processing speed of image replacement by each method must be faster than that of sprite drawing. Therefore, in this study, we test the processing speeds of the software based version and the hardware based version of the image replacement method. Then, we evaluate whether it can be hidden in the processing time of sprite drawing.

4. Experiment and Discussion

4.1 Experiment Environment

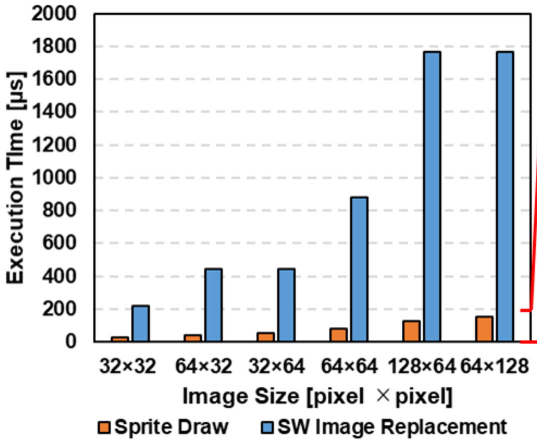
We used a high-level synthesis tool of Xilinx Vitis HLS 2022.2. The generated HDL programs were simulated and converted into circuit data for writing to an FPGA by Xilinx's FPGA development environment software, Vivado 2022.2. The FPGA is a Xilinx Zynq XC7Z020, and the FPGA bord is PYNQ -Z1 from DIGILENT. The tests were performed on its FPGA board. On the FPGA board used, the operating frequency of the embedded CPU on the FPGA is 650 MHz and that of the FPGA is 100 MHz. To test the efficiency of proposed image replacement systems, we use several size of sprite images that is 32×32, 64×32, 64×32, 64×64, 128×64 and 64×128 (width×height).

4.2 Experimental Result and Discussion

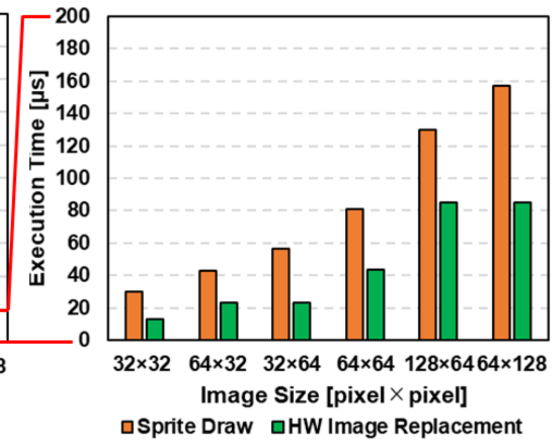
To verify whether the image replacement time can be hidden in the processing time of sprite drawing, this paper tests and compares the execution time of each image replacement system and sprite drawing. Fig.7 shows the results of the execution time comparison.

Fig. 7(a) is the comparison processing time of the sprite drawing and the software-based image replacement. As can be seen in this figure, the execution time of the software-based version is much longer than that of the sprite drawing. Therefore, it was found that software-based image replacement time could not be hidden by sprite drawing.

In contrast, Fig. 7(b) shows the results of a comparison of the processing times of the sprite drawing and the hardware-based image replacement. As can be seen in this figure, the execution time of hardware-based version is faster than that of sprite drawing for the same size image. Therefore, as shown in Fig. 6, the processing time of hardware-based image replacement can be hidden by that of

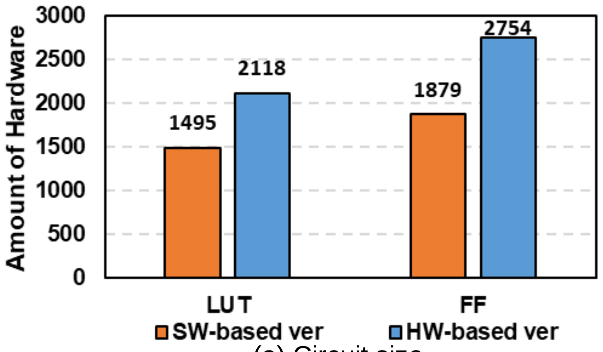


(a) Software-based version.

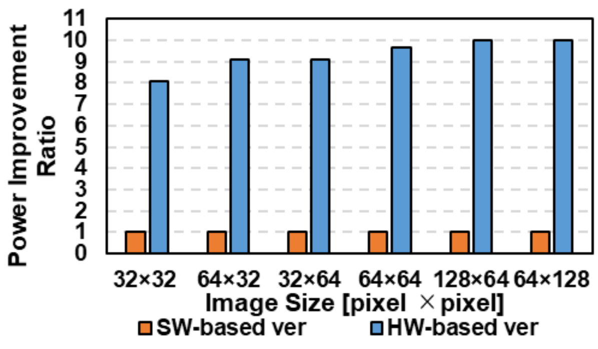


(b) Hardware-based version.

Fig. 7. Comparison with sprite drawing.



(a) Circuit size.



(b) Power improvement ratio.

Fig. 8. Comparison SW-based version and HW-based version.

sprite drawing. Thus, it is possible to increase the number of images that can be handled while keeping the actual processing time the same.

Based on the above, we compared the circuit size and power improvement ratio between the software-based and hardware-based systems. The results are shown in Fig. 8. Fig. 8(a) shows the circuit size and Fig. 8(b) compares the power improvement ratio calculated based on the execution time and circuit size. The circuit size is the sprite drawing hardware plus the amount of DMAC hardware in the case of the hardware-based system. The power improvement ratio is compared at each image size, where the software-based version is set to 1 respectively. As can be seen from Fig. 8(a), the hardware-based version has a larger circuit size due to the DMAC circuitry required. However, the software-based version takes significantly longer to replace, so the processing time is much longer than the hardware-based version. Therefore, despite the larger circuit size, the power efficiency of the hardware-based version is more than 8 times better than that of the software-based version, as shown in Fig. 8(b).

The above shows that the hardware-based system is suitable for the image replacement process that is performed in the background of the process.

5. Conclusion

This paper proposed the image replacement system for sprite drawing hardware that uses the FPGA's internal memory and evaluated the software-based and hardware-based versions of the system.

As a result, it was found that the software version was considerably slower in processing time than sprite drawing. In contrast, the hardware version was revealed to be faster than sprite drawing for the same image size, and its replacement time could be hidden. Therefore, the hardware-based version of the image replacement system can increase the number of images that can be handled with the same processing time as before.

In the future, we would like to evaluate the random-access performance of the FPGA internal memory and develop hardware for different sprite processing, such as sprite rotation and enlargement.

References

[1] Y. Yamagata and A. Yamawaki, " A Consideration to Develop a High-level Synthesizable Software Game Library", ***Proceedings of the 6th IIAE International Conference on Industrial Application Engineering 2018***, pp.202-205, 2018.

[2] F. Muslim, L. Ma, M. Roozmeh and L. Lavagno " Efficient FPGA Implementation of OpenCL High-Performance Computing Applications via High-Level Synthesis", ***IEEE Access***, Vol.5, pp.2747-2762, 2017.

[3] R. Nane, V. Sima, B. Olivier, R. Meeuws, Y. Yankova and k. Bertels, " DWARV 2.0: A CoSy-based C-to-VHDL hardware compiler", ***Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL)***, pp.619-622, 2012.

[4] D. Shirai and A. Yamawaki, “Hardware Development of Skylight Estimation Processing in Haze Removing Using High-level Synthesis”, ***Proceedings of the 9th IIAE International Conference on Industrial Application Engineering 2021 (ICIAE2021)***, pp.107-111, 2021.

[5] J. Qin and A. Yamawaki, “Hardware Development of Edge-Preserving Bubble Image Conversion in High-level Synthesis”, ***Proceedings of the International Conference on Artificial Life and Robotics***, pp.608-611, 2022.

[6] K. Lee and A. Yamawaki, “Background scrolling in high-level synthesis oriented game programing library”, ***Artificial Life Robotics,*** Vol. 27, pp.455-460, 2022.

[7] H. Tani and A. Yamawaki, “Effect of Line Segment Size on Pencil Drawing-like Image Conversion Hardware Developed by High-level Synthesis”, ***Proceedings of the 28th International Symposium on Artificial Life and Robotics 2023 (AROB 2023)***, pp740-743, 2023.

[8] K. Kurata and A. Yamawaki, “Hardware Development of Sphere Intersection in Ray-casting Using High-Level-Synthesis”, ***Proceedings of the 7th Imaging, Signal Processing and Communications 2023 (ICISPC2023)***, pp.86-90, 2023.

[9] K. Matsuda and A. Yamawaki, “Investigation Toward Realization of High-level Synthesizable Sprite Moving Functions”, ***Proceedings of the 11th IIAE International Conference on Industrial Application Engineering 2023***, pp.269-275, 2023.

[10] Y. Otani and A. Yamawaki, “Development of High-performance Sprite Drawing Process in a Game Library for High-Level Synthesis”, ***Proceedings of the 2023 5th International Conference on Computer Communication and the Internet (ICCCI)***, pp.223-228, 2023.

[11] K. Aoki and A. Yamawaki, “Development of Sprite Drawing Hardware Combining High-Level Synthesis and FPGA Internal Memory”, ***Proceedings of the 2024 6th International Electronics Communication Conference***, pp.37-41, 2024.