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

Development of single frame sprite drawing hardware in High-level Synthesis

Yuka Otani[1, a] and Akira Yamawaki[1,b, *]

[1] Kyushu Institute of Technology, 1-1 Sensui-cho, Tobata-ku, Kitakyushu-shi, Fukuoka 804-8550, Japan

[a]< ootani.yuuka556@mail.kyutech.jp >, [b]< yama@ecs.kyutech.ac.jp >

Keywords: high-level synthesis, fpga, sprite, drawing, game libraries

Abstract. As part of a hardware-oriented game library created using high-level synthesis, development of hardware capable of drawing multiple sprites in a single frame was addressed. The step of drawing sprites in a single frame is effectively achieved by filling one frame with a transparent color and overwriting that frame with the multiple sprites required for the next frame. In this paper, we develop hardware to realize sprite drawing in a single frame using high-level synthesis. The performance and usefulness of the proposed hardware is verified through experiments using actual devices.

1. Introduction

Field programmable gate array (FPGA) based mobile terminals can implement applications with higher performance and power savings by hardware execution of computationally demanding processes. We are working on the development of a game library optimized for high-level synthesis (HLS) [1-2], a technology that automatically converts software suitable for developing such applications to hardware [3-7]. Among them, we conducted basic research on the drawing of sprites (small 2 Dimension (2D) characters) used in 2D games [8-10].

Those previous studies were basic research and development on rendering individual sprites. However, in general, there are many sprites in a game. In this study, we propose a drawing algorithm that seems suitable for drawing many sprites and developing hardware-oriented software description for it. Then, we prove the effectiveness of the proposed method through experiments on FPGA implementation. Then, we prove the effectiveness of the proposed method through experiments on FPGA implementation.

2. Multiple Sprite Drawing

A game screen is composed of multiple screens, including background and sprite screens to realize . an emotional atmosphere. This study focuses on one of the sprite screens and considers how to realize a drawing with multiple moving sprites.

Fig. 1 shows a conceptual diagram of the proposed algorithm for drawing multiple sprites. To realize the movement of multiple sprites on a sprite screen, the sprite screen is filled with a transparent color, and the sprites are drawn on it in order of priority, starting with the sprite with the lower priority.

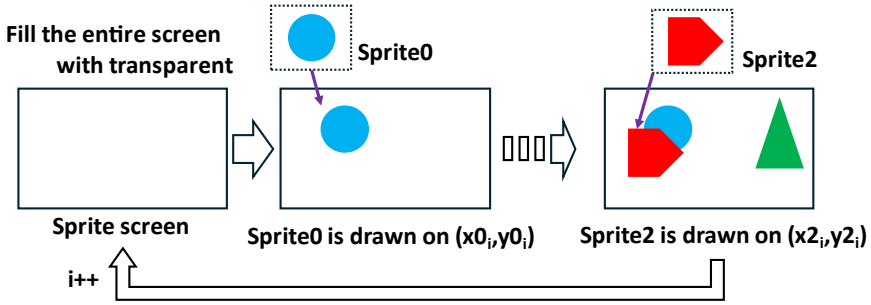
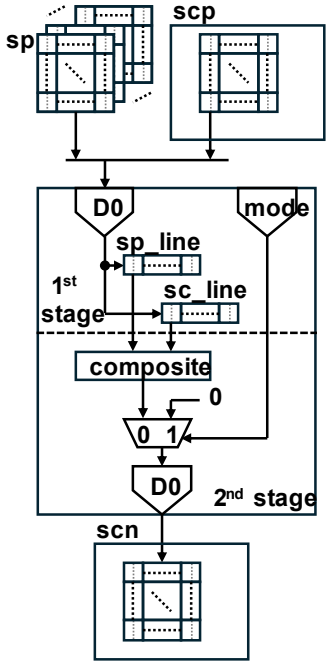


Fig. 1. Concept of multiple sprite drawing.



(a) Hardware organization

```
1:ReadData(u64 *sp,*scp,*sl,*scl, i32 i, u16 x,y,w, u8 mode) {
2:  if( mode == 1 ) return;
3:  L11:for(j=0;j<(w/2);j++){ sl[j] = sp[i*(w/2)+j];}
4:  L12:for(j=0;j<(w/2);j++){ scl[j]= scp[(i+y)*(GAME_DW/2)+(x/2)+j];}}
5:void SpriteDrawHW( u64 *sp,*scp, u16_t x,y,h,w, u64 *scn, u8 mode ){
6:#pragma HLS ... M_axi bundle=d0 port=sp
7:#pragma HLS ... M_axi bundle=d0 port=scp
8:#pragma HLS ... M_axi bundle=d0 port=scn
9:  u64_t sl[MAX_SPW / 2];
10:#pragma HLS...sl type=ram_2p impl=lutram
11:  u64_t scl[MAX_SPW / 2];
12:#pragma HLS...scl type=ram_2p impl=lutram
13:  L1: for(i=0;i<h;i++){
14:#pragma HLS DATAFLOW
15:    ReadData(sp,scp,sl,scl,i,x,y,w,mode);
16:    L13: for(j=0;j<(w/2);j++){
17:      if( mode == 1 ) px[5] = px[4] = 0;
18:      else { px[0] = sl[j];  px[1] = sl[j]  >> 32;
19:             px[2] = scl[j]; px[3] = scl[j] >> 32;
20:             if( pix[0] != 0 ) px[4] = px[0];
21:             else              px[4] = px[2];
22:             if( pix[1] != 0 ) px[5] = px[1];
23:             else              px[5] = px[3];  }
24:      scn[(i+y)*(GAME_DW/2)+(x/2)+j] = (u64)px[5] << 32 | (u64)px[4];
25:}}}
```

(b) Proposed software description

Fig. 2. Proposed method.

When the drawing of a single screen is completed, the sprite screen is again filled with transparent color and each group of sprites is redrawn at the next coordinate, in accordance with the frame rate timing. By repeating this process, multiple sprites moving can be realized. This method is called single-frame sprite drawing because the process proceeds in units of one sprite screen.

3. Hardware-Oriented Single Frame Sprite Drawing

3.1 Proposed Hardware Organization and Software Description

This paper develops a hardware module erasing a whole sprite screen and drawing a single sprite to make the coordinates and number of sprites arbitrarily. Fig. 2(a) and (b) show the organization of the proposed hardware and software description. In Figure 2, **sp** is the group of sprite images, **scp** is the previous sprite screen, and **scn** is the sprite screen after processing.

The first stage of hardware in (a) reads one line of the target sprite image and one line at the concerned position on the sprite screen into the line buffers. The second stage combines the contents of the line buffers and updates the sprite screen. The 64 bits input/output ports can access and process 2 pixels simultaneously.

In (b), **SpriteDrawHW** is the hardware target function. The first stage in (a) corresponds to the **ReadData** function, and the second stage to loop L13. Each stage is realized in parallel by the DATAFLOW pragma in line #14. However, the reading of the sprite image (L11) and the sprite screen (L12) in **ReadData** is performed sequentially. Therefore, both reading and writing can be assigned to a single AXI bus (D0). The functions of erasing and drawing are toggled by the input “mode” (“1” for erasing, “0” for drawing). Then, the pragmas are used in line #10 and #12 to reduce the size of the circuit by using a LUT instead of a BRAM.

3.2 Parallel Execution of Proposed Hardware

To achieve faster and high performance operation using the proposed hardware, it is conceivable that not a single unit of developed drawing hardware, but two, three or more units could be installed. This is shown in Fig. 3.

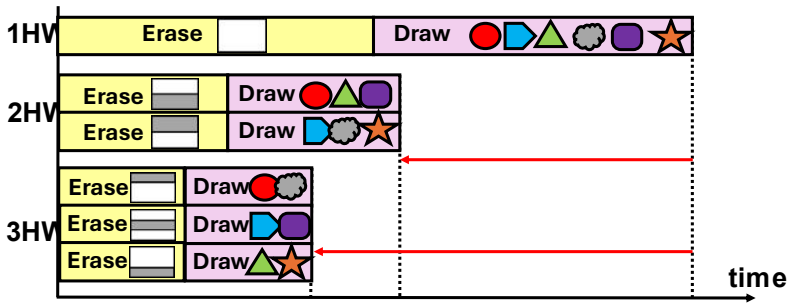
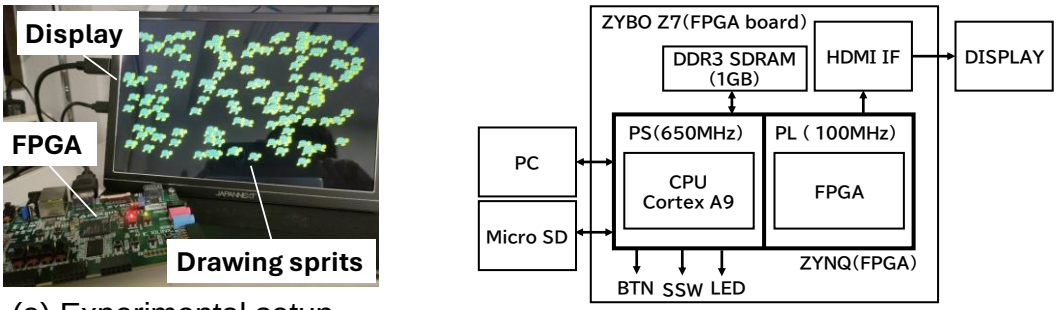


Fig. 3. Parallel execution of proposed hardware.



(a) Experimental setup (b) Hardware platform

Fig. 4. Experimental environment.

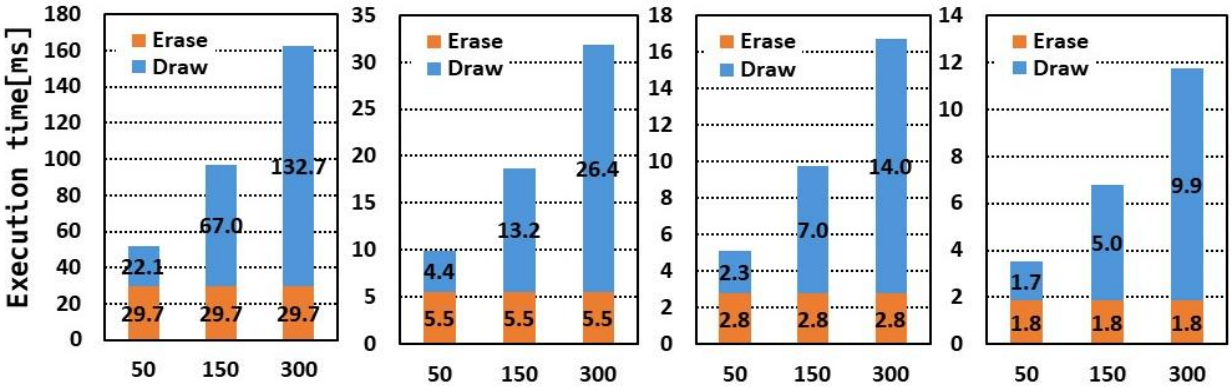
With one drawing hardware, it is necessary to draw one sprite at a time, which would fill all the screen with a transparent color and draw several sprites. A single drawing hardware needs to fill all of one screen with a transparent color and draw multiple sprites one at a time. However, if multiple units of hardware are installed, it is possible to share the work of filling a single screen and to draw multiple sprites at the same time, as many as the number of hardware units installed at one time. Therefore, we considered that the more sprites that are drawn in a frame, the more speed-up can be achieved. In this study, this experiment is conducted by varying the number of implementation drawing hardware and the number of drawing sprites.

4. Experiments and Discussions

The developed testbed system is shown in Fig. 4(a). This system is built on a Digilent ZYBO Z7-10 FPGA board. The board is equipped with a Xilinx ZYNQ FPGA, XC7Z10-1CLG400. This testbed system has a display connected by the high-definition multimedia interface (HDMI). Using the display, it is possible to visually check how the sprites move. Fig. 4(b) shows a visual test being performed. Multiple turtle sprites are drawn all over the screen. The sprite image size is 64x64 and the screen size is 1280x720. The number of sprites drawn per frame varied from 50, 150 and 300 pieces. Each sprite is drawn at random coordinates to achieve a casual trajectory. The HLS tool used to generate the hardware modules was Xilinx Vitis HLS 2022.2. The hardware module was implemented with Xilinx Vivado 2022.2. The embedded central processing unit (CPU) is the ARM Cortex A9 at 650MHz. The programmable logic, PL, which configures the HLS hardware modules runs at 100 MHz.

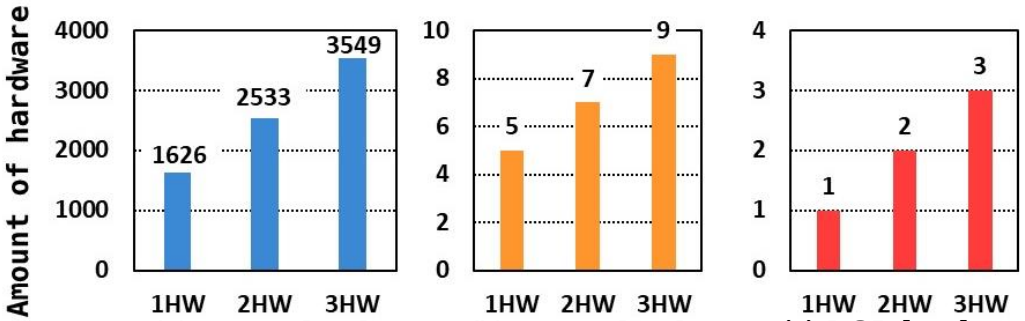
4.1 Execution Time

The time taken to erase the entire screen and draw the specified number of sprites was measured. To improve the accuracy of the measurement results, 100 frames were drawn and the average execution time per frame was calculated. The results are shown in Fig. 5. As a measurement of single frame processing, the time required for erasing and drawing is shown as a detailed breakdown of the time for each. (a) is the result of software execution by an embedded CPU (SW), (b) is the result of hardware execution using one sprite drawing hardware (1HW), (c) is the result of hardware execution using two (2HW), and (d) is the result of hardware execution using three (3HW).



(a) SW (b) 1HW (c) 2HW (d) 3HW

Fig. 5. Execution time.



(a) SLICE [pcs] (b) BRAM [pcs] (c) DSP [psc]

Fig. 6. Amount of hardware.

The figure shows that 1HW reduced the execution time by about 80.6% compared to SW. For the results of hardware execution, we focus on the results for each number of units installed. Compared to 1HW, it was confirmed that 2HW reduced execution time by about 48.1% and 3HW reduced by about 63.7%. However, focusing on the drawing time when the 150 pieces are drawn. Since the result in (b) was 13.2 ms, we assumed that each result would be 6.6 ms in (c) and 4.4 ms in (d) according to the number of drawing hardware units. The actual results were 7.0 ms for (c) and 5.0 ms for (d). The more the number of sprites drawn, the more time than estimated value is taken. This can be assumed to be due to the increase in access waiting time to the shared DDR3 SDRAM shown in Fig. 4(b).

4.2 Amount of Hardware

Fig. 6 shows the circuit size for each resource when the number of installed hardware is changed. (a) shows the amount of resources of SLICE, (b) of BRAM, and (c) of DSP.

Based on the results in (a), compared to 1HW, 2HW is about 1.6 times and 3HW is about 2.2 times larger. The breakdown of SLICE for the overall hardware was focused on. The number of 1HW was 883, 2HW was 1690, and 3HW was 2633. So, when focusing on the drawing hardware, the increase was regular in relation to the number of hardware units installed. Therefore, it is thought that the overall rate of increase may have been slower because shared resources were used in the wrapper part for embedding the drawing hardware, such as the connection to each block element.

Then, focusing on BRAM, 2HW was 1.4 times larger than 1HW, and 3HW was 1.8 times larger than 1HW. BRAM is a memory implemented inside the FPGA. It was used three for the interface for the display and two for each drawing hardware. Therefore, it can be said that the increase is regular and in relation to the number of units installed.

Focusing on DSP, 2HW was 2.0 times larger than 1HW, and 3HW was 3.0 times larger than 1HW. It is thought that this is necessary to calculate the array index for each drawing hardware. Therefore, the number of DSP is provided for the number of units installed.

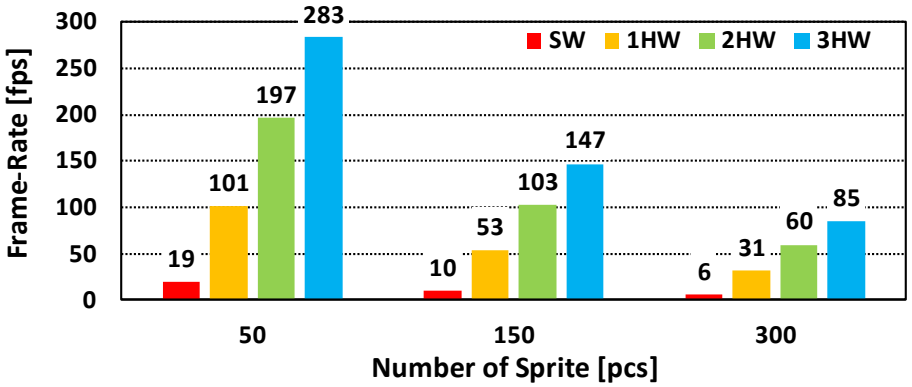


Fig. 7. Frame rate.

4.3 Frame-Rate

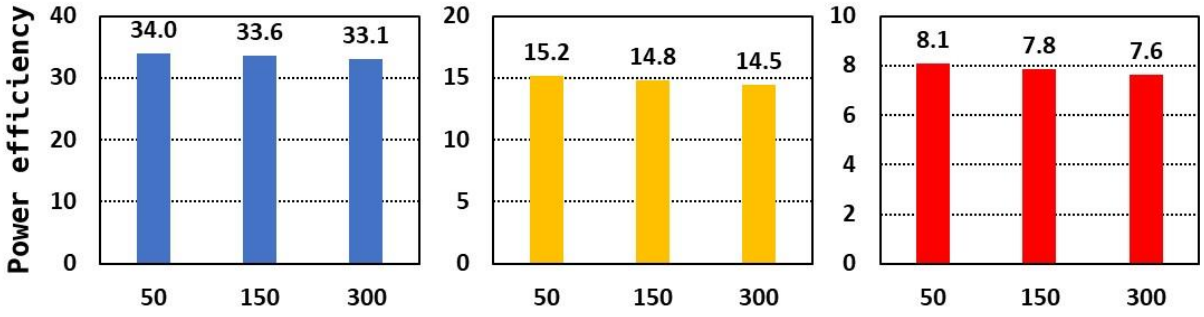
Fig. 7 shows the results of the frame rate that is calculated from the execution time in **4.2**. The frame rate is an indicator of how many frames are displayed per second. The larger the value, the smoother the motion. The figure shows that when the number of drawn sprites increased, the frame rate decreased. This is because the processing time required for drawing increases. However, all hardware executions achieved higher frame rates than software executions by a factor of about 5.0 or more. Then, comparing the frame rate results with 1HW, 2HW was about 1.9 times higher and 3HW was about 2.8 times higher. The reason for this is that the more drawing hardware is installed, the faster a single frame can be completed.

4.4 Power Efficient

As the circuit size increases, power consumption increases accordingly. The power improvement ratio was calculated using eq. (1). The circuit size in the case of 1HW is used as the standard and the rate of increase of the circuit size is considered by multiplication of the individual degrees of increase or decrease individually.

$$power\ effciency = \frac{SW_{execution\ time} \times SW_{operating\ frequency}}{HW_{execution\ time} \times HW_{operating\ frequency} \times \frac{SLICE}{SLICE_{1HW}} \times \frac{BRAM}{BRAM_{1HW}} \times \frac{DSP}{DSP_{1HW}}} \quad (1)$$

The power improvement ratio to software execution is shown on Fig. 8. The horizontal axis is used as the number of drawing hardware [pcs] and the vertical axis is the power improvement ratio. The figure shows that 1HW improved the power by about 33.5 times, 2HW by about 14.8 times and 3HW by about 7.9 times. The 3HW had the fastest execution time for a single frame, thus achieving a high frame rate. However, it required more circuit resources and had the lowest power improvement ratio of the number of mounted hardware units measured. In contrast, 1HW had the longest execution time and lowest frame rate values among the hardware executions. However, the circuit size was the smallest and the power improvement ratio was 4.3 times higher than that of 3HW.



(a) 1HW (b) 2HW (c) 3HW

Fig. 8. Power efficiency.

The results are discussed when 300 sprites are drawn, which Fig 7 and Fig 8 show. The 1HW had a frame rate of 31 fps, the lowest result among the number of hardware installations. In contrast, 3HW had the highest result of 85 fps. The results of the power improvement ratio showed that 1HW had the highest value and 3HW had the lowest value. As Fig. 6 shows, this is because the amount of resources required increases as the number of on-board units increases.

Accordingly, lower frame rates require lower power. On the other hand, higher frame rates require higher power. From the above, these two items can be said trade-off relationship.

5. Conclusion

In this paper, hardware for drawing a large number of sprite images on a sprite screen was developed using high-level synthesis. As a result, the hardware achieves faster processing than software. In addition, by simultaneously running multiple units of drawing hardware, it was possible to realize even higher speeds and higher frame rates. Then, it was found that low power consumption requires executing at a low frame rate, and high frame rate requires high power consumption. From this, it can be said that frame rate and power are a trade-off. Therefore, it is necessary to make a choice according to the need.

As a future work, we would like to develop drawing hardware that can composite not only sprite frames but also a background.

References

[1] F. B. Muslim, L. Ma, M. Roozmeh, L. Lavagno, "Efficient FPGA implementation of OpenCL high-performance computing applications via high-level synthesis", ***Journal of IEEE Access***, vol 5, pp. 2747-2762, 2017.

[2] F. Ferrandi et al., " Invited: Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications", ***Proc. of DAC2021*** (San Francisco, USA), November 2021.

[3] Y. Yamagata and A. Yamawaki, "A Consideration to Develop a High-level Synthesizable Software Game Library", ***Proc. of ICIAE2018*** (Naha, Japan) March 2018.

[4] J. Qin and A. Yamawaki, "Hardware Development of Edge-Preserving Bubble Image Conversion in High-level Synthesis", ***Proc. of ICAROB2022***, January 2022.

[5] K. Lee and A. Yamawaki, "Background scrolling in high-level synthesis oriented game programing library", ***Journal of Artificial Life and Robotics***, Vol 27, Issue 3, pp.455-460, 2022.

[6] H. Tani and A. Yamawaki, "Effect of Line Segment Size on Pencil Drawing-like Image Conversion Hardware Developed by High-level Synthesis", ***Proc. of AROB2023*** (Beppu, Japan), January 2023.

[7] K. Lee and A. Yamawaki, "An investigation of software describing methods to design dual background scrolling hardware in high-level synthesis", ***Journal of Artificial Life and Robotics***, Vol 28, Issue 3, pp 547-552, 2023.

[8] Y. Otani and A. Yamawaki, "Data Path parallelization to Improve Performance of High-level Synthesized Sprite Drawing Hardware", ***Proc. of ICISPC2023*** (Kumamoto, Japan), July 2023.

[9] Y. Otani and A. Yamawaki, "Performance variation caused by sprite drawing pattern for high-level synthesized sprite drawing hardware", ***Proc. of IECC2024*** (Fukuoka, Japan), July 2024.

[10] Y. Otani and A. Yamawaki, " Development of sprite drawing hardware with two parallelisms using semi-parallel method", ***Proc. of ICISIP2024*** (Matsuyama, Japan), September 2024.