A Consideration to Develop a High-level Synthesizable Software Game Library

Yuki Yamagata and Akira Yamawaki*

Kyushu Institute of Technology, 1-1 Sensui, Tobata, Kitakyushu 804-8550, Japan

*Corresponding Author: yama@ecs.kyutech.ac.jp

Abstract

To reduce power consumption of mobile device for extending the battery life, an efficient architecture of the mobile device is required in addition to improve the battery capacity. The hardware realization of the high computational load is one of the promised methods to reduce the power consumption significantly. Especially the dynamic reconfiguration of hardware on a reconfigurable device is seemed to be a suitable technology to meet the mobile device nature which executes many kinds of applications dynamically. We plan to develop a high-level synthesizable software game library so that the game application can be executed on CPU and reconfigurable hardware. The HLS tool tends to generate a dully circuit when the pure software is described without consideration of the hardware organization. This paper attempts to make an efficient software program drawing the rectangular on the display which is generally included in any game library.

Keywords: high-level synthesis, game library, reconfigurable device, mobile device, low power.

1. Introduction

To reduce power consumption of mobile device for extending the battery life, an efficient architecture of the mobile device is required in addition to improve the battery capacity. The hardware realization of the high computational load is one of the promised methods to reduce the power consumption significantly ⁽¹⁻³⁾. However, developing all custom hardware modules executing the mobile applications on application store are impossible. The dynamic hardware reconfiguration ⁽⁴⁻⁵⁾ on a reconfigurable device like FPGA is seemed to be a suitable technology to meet the mobile device nature executing many kinds of applications dynamically.

The game application is one of power-hungry mobile applications. In addition, the game applications are most popular for the mobile device users. Thus, it is seemed to be significant to develop the mobile device supporting the game application that can be executed on the reconfigurable device.

There are many pure software game libraries ⁽⁶⁻⁸⁾ to make game development efficient. However, the game software libraries that the high-level synthesis (HLS) ⁽⁹⁻¹¹⁾ can convert to the hardware automatically have never seen before. The HLS is a promising technology to make hardware development easy and quick. This nature is suitable for the mobile applications that the new comers frequently appear. Thus, we think to develop a high-level synthesizable software game library, HGL, is meaningful.

Before developing the full HGL, this paper investigates the efficient hardware descriptions at C language level so that the HLS tool can generate the high performance hardware module. As first step, we focus on the drawing rectangular which is one of the common functions across the pure software game libraries ⁽⁶⁻⁸⁾. We attempt HLS tool to convert the software function drawing the rectangular to the hardware supporting bus burst transaction connected with the pipelined data path. Such hardware module can achieve the high performance by generating pixels of the rectangular continuously into the memory.

The rest of paper is organized as follows. Section 2 describes a conceptual architecture of the mobile device supporting dynamic hardware reconfiguration. Section 3 shows the describing method of the rectangular function which the HLS tool may convert to the hardware module supporting the bus burst transfer, compared with the pure software function of the rectangular. Section 4 performs the performance evaluation to clarify the effectiveness of the rectangular function we describe. Finally, Section 5 concludes this paper and indicates the future work.

Fig. 1. Concept of Mobile Device with Reconf. HW

2. Mobile Device with Reconf. Hardware

2.1 Content Delivery Service and Mobile Device

The mobile device can download any kind of applications from the content delivery services like Google play, App store and so on. Then, we enjoy applications downloaded on our mobile device. Using the mobile device with reconfigurable hardware, this concept should be same seamlessly.

Fig. 1 depicts a concept of the content delivery service and mobile device with reconfigurable hardware executing the downloaded applications. The content delivery service holds the applications consisting of the software code and hardware code in addition to the conventional software applications including only software code. The software code is the traditional executable code of the processor. The hardware code is the circuit data which configures the hardware module on the reconfigurable parts.

Fig. 2. Conceptual Flow of Application Development

The mobile device with reconfigurable hardware, MDRH, mainly consists of the processor, hardware reconfigurable part, and some storage. The reconfigurable part has one or more reconfigurable partitions which can implement multiple hardware modules simultaneously. Each reconfigurable partition can be dynamically reconfigured to other hardware modules while the MDRH is running.

The software code is executed by the processor. The hardware code configures the hardware module on the empty reconfigurable partition. If there is no empty reconfigurable partition, the hardware module on a reconfigurable partition at idle state is replaced. The applications can be switched as traditional task switching. The hardware module in the application to be switched can remain on the reconfigurable part. So, the task switching overhead can be eliminated when the application switched is re-executed, which is using the remaining hardware module on the reconfigurable partition.

2.2 Development Flow of Application

The compiler for mobile device with reconfigurable hardware has to include the HLS engine converting the software to the hardware circuit data in addition to the traditional software compile engine like GCC as shown in Fig. 2.

The application developer describes the source code in a high level language like C, C++, JAVA, and so on. The parts to be converted to the hardware modules are specified by the functions marked. The compiler compiles the source code replacing the hardware function to the managing and communicating instructions to the hardware modules on the reconfigurable partitions. Then, the software compiler compiles the software source code reconstructed into the executable binary code on the processor. The HLS engine converts the hardware function into the hardware behavior in the hardware description language, HDL, performs the logic synthesis to the HDL program converted, and executes the place and route for the real reconfigurable part to generate the circuit data.

3. Rectangular Function for HLS

3.1 Pure Software

Fig. 3 shows a pseudo code in C to realize the drawing rectangular. The argument of **f** is the pointer to the frame memory to be displayed. The arguments of **x1** and **y1** are the coordinates of the rectangular origin. The arguments of **x2** and **y2** are the coordinates of the rectangular end point. The

```

1: // DW is display width
2: DrawRect(*f,x1,y1,x2,y2,col,f1){
3:   if(f1){ //filled rect
4:     for(y=y1*DW;y<=y2*DW;y+=DW)
5:       for(x=x1;x<=x2;x++) f[x+y]=col;
6:   }else{ //blanked rect
7:     for(y=y1*DW;y<=y2*DW;y+=DW){
8:       if(y==y1*DW || y==y2*DW)
9:         for(x=x1;x<=x2;x++) f[x+y]=col;
10:      else f[x1+y]=f[x2+y]=col;
11:    }
12:  }
13: }

```

Fig. 3. Pure Software Function of Rectangular

argument of **col** is the color to be filled with in the rectangular. The **f1** is the flag indicating whether the rectangular is filled with the specified color or not.

When the **f1** is set to 1, the rectangular filled with the pixel colored by **col** is drawn in the frame memory. When the **f1** is set to 0, the blanked rectangular is drawn in the frame memory. The blanked rectangular is circled by the lines colored by **col**.

3.2 Software Description Tailored to HLS

Fig. 4 shows the rectangular function reconstructed for HLS to infer the bus burst transfer. The arguments of this function are same as Fig. 3. To make HLS infer the bus burst transfer, we describe the specified loops denoted as **FL3** and **BL3**. These loops perform memory store operations repeated by the burst length, **BL**. The remaining pixels when the

```

1: // DW is display width
2: // BL is burst length
3: HLS_DrawRect(*f,x1,y1,x2,y2,col,f1){
4:   if(f1){ //filled rect
5:     FL1:for(i=y1;i<=y2;i++){
6:       idx_i=DW*i+x1;
7:       FL2:for(j=0;j<(x2-x1+1)/BL;j++){
8:         idx_j=idx_i+j*BL;
9:         // to infer burst transfer
10:        FL3:for(k=0;k<BL;k++)f[idx_j+k]=col;
11:      }
12:      idx_i+=((x2-x1+1)/BL)*BL;
13:      // draw the remaining pixels
14:      FL4:for(j=0;j<(x2-x1+1)%BL;j++){
15:        f[idx_i+j]=col;
16:      }
17:   }else{ // blanked rect
18:     BL1:for(i=y1;i<=y2;i++){
19:       idx_i=DW*i+x1;
20:       if(i==y1||i==y2){
21:         BL2:for(j=0;j<(x2-x1+1)/BL;j++){
22:           idx_j=idx_i+j*BL;
23:           // to infer burst transfer
24:           BL3:for(k=0;k<BL;k++)f[idx_j+k]=col;
25:         }
26:         idx_i+=((x2-x1+1)/BL)*BL;
27:         // draw the remaining pixels
28:         BL4:for(j=0;j<(x2-x1+1)%BL;j++){
29:           f[idx_i+j]=col;
30:         }
31:       }else f[idx_i]=f[DW*i+x2]=col;
32:     }
33:   }

```

Fig. 4. Rectangular Function for HLS

Fig. 5. Experimental Hardware Organization

number of pixels is divided by the burst length are also stored into the frame memory at the loops of **FL4** and **BL4**. Thus, this function can handle any number of pixels even when the number of pixels cannot be divided by the burst length.

4. Experiment and Discussion

4.1 Experimental Setup

To perform the experiment on the real machine, we have developed the hardware platform shown in Fig. 5. This platform has the wrapper, **HLS_wrap**, covering the HLS hardware, **HLS_HW**. Due to this wrapper, the HLS hardware is replaceable on the platform easily.

The hardware platform has been built on Xilinx ZYNQ FPGA which is equipped on the FPGA board, Digilent ZYBO. The embedded processor of ARM Cortex-A9 runs at 650MHz. The hardware modules run at 100MHz.

To perform comparative evaluations among the software and hardware execution, the software was compiled by GCC with **-O2** on the Xilinx SDK launched from Vivado 2016.4. The Vivado HLS 2016.4 was used to perform HLS. The VHDL programs generated by the Vivado HLS are compiled by the Vivado 2016.4 to generate the circuit data stream. To measure the execution time of the software and hardware, we have equipped the performance counter running at 100MHz as a memory mapped register on the FPGA.

Before HLS, we have inserted pragmas for the pipelining to appropriate loops. In addition, we have turned off the loop flatten for some loops that make the hardware size significant large regardless of no performance improvement.

4.2 Result and Discussion

Fig. 6 shows the measured execution time. The **PSW** is the execution time of pure software shown in Fig. 3 on the embedded processor. The **HW_nb** is that of HLS hardware generated from the pure software shown in Fig. 3 with the pragmas. The **HW_B16** means that of the HLS hardware generated from the restructured software shown in Fig. 4

Fig. 6. Measured Execution Time

with pragmas for HLS tool to infer the bus burst transfer with 16 burst length.

Fig.6 shows that our reconstruction to the pure software can improve the performance of 3.8 times compared with the software execution by inferring the bus burst transfer. Also, the hardware generated from the pure software cannot overcome the performance of software execution. This fact indicates that the deep consideration of the hardware nature at C software level is very important to make HLS tool generate well-structured hardware module.

5. Conclusions

To extend the battery life of the mobile devices, the hardware realization of the high computational load is one of the promised methods to reduce the power consumption significantly. The dynamic reconfiguration of hardware on a reconfigurable device is seemed to be a suitable technology to meet the mobile device nature which executes many kinds of applications dynamically.

The HLS is a promising technology to make hardware development easy and quick. The game applications are most popular for the mobile device users. There are many pure software game libraries to make game development efficient. However, the game software libraries that the HLS tool can convert to the hardware automatically have never seen before. Thus, we think to develop a high-level synthesizable software game library, HGL, is meaningful.

Before developing the full HGL, this paper investigates the describing method of the drawing rectangular function in C so that the HLS tool can generate the high performance hardware module. The experimental result shows the effectiveness of our describing method can improve performance significantly compared with the pure software description by inferring the bus burst transfer.

As future work, we will develop more high-level synthesizable functions of the game library. Also, we will develop real game using our HLS game library.

References

- (1) Robert Karam, Ruchir Puri, and Swarup Bhunia, "Energy-Efficient Adaptive Hardware Accelerator for Text Mining Application Kernels", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 24, No. 12, pp. 3526-3537, 2016.
- (2) Shih-Hsiang Lin, Pei-Yin Chen, and Yu-Ning Lin, "Hardware Design of Low-Power High-Throughput Sorting Unit", *IEEE Transactions on Computers*, Vol. 66, No. 8, pp. 1383-1395, 2017.
- (3) J. M. Mora-Gutiérrez, C. J. Jiménez-Fernández, and M. Valencia-Barrero, "Multiradix Trivium Implementations for Low-Power IoT Hardware", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 25, No. 12, pp. 3401-3405, 2017.
- (4) Anju P. Johnson, Sikhar Patranabis, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay, "Remote dynamic partial reconfiguration: A threat to Internet-of-Things and embedded security applications", *Microprocessors and Microsystems*, Vol. 52, pp. 131-141, 2017.
- (5) S. Navid Shahrouzi and Darshika G. Perera, "Dynamic partial reconfigurable hardware architecture for principal component analysis on mobile and embedded devices", *EURASIP Journal on Embedded Systems* 2017:25, <https://doi.org/10.1186/s13639-017-0074-x> 2017.
- (6) Takumi Yamada, <http://dxlib.o.oo7.jp/>, [Accessed: 15-Feb-2018] in Japanese.
- (7) GitHub Inc., "Play Siv3D! A free C++ library for games and interactive media", <http://play-siv3d.hatenablog.com/>, [Accessed: 15-Feb-2018].
- (8) Laurent Gomila, "Simple and Fast Multimedia Library", <https://www.sfml-dev.org/index.php>, [Accessed: 15-Feb-2018]
- (9) R. Nane, V.-M. Sima, B. Olivier, R. Meeuws, Y. Yankova, and K. Bertels, "DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler", In *FPL*, pp. 619-622. 2012.
- (10) C. Pilato and F. Ferrandi, "Bambu: A Modular Framework for the High Level Synthesis of Memory-intensive Applications", In *FPL*, pp. 1-4, 2013.
- (11) A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J.H. Anderson, S. Brown, and T. Czajkowski, "LegUp: High-Level Synthesis for FPGA-based Processor/Accelerator Systems", In *ACM FPGA*, pp. 33-36. 2011.